

ALMA MATER STUDIORUM · UNIVERSITÀ DI
BOLOGNA CAMPUS DI CESENA

SCUOLA DI SCIENZE
Corso di Laurea in Ingegneria e Scienze informatiche

Previsione di agenti inquinanti mediante
reti neurali e ottimizzazione degli
iperparametri attraverso grid search con
Talos

Relazione finale in:

SYSTEMS INTEGRATION

Relatore:
Prof. VITTORIO GHINI

Presentata da:
GIACOMO TONTINI

Seconda Sessione di Laurea
Anno Accademico: 2017-2018

ringraziamenti

Giunti al termine di questo percorso vorrei ringraziare in primo luogo la mia famiglia, che mi ha sempre sostenuto e sopportato nei momenti più difficili oltre ad aver reso possibile che tutto ciò si realizzasse nel migliore dei modi.

Vorrei ringraziare il professor Vittorio Ghini, che oltre ad essere il relatore di questo progetto di tesi, è sempre stato disponibile a dedicarmi il suo tempo ed è stato fonte di supporto e motivazione.

Meritano di essere ringraziati i miei compagni di avventura, Daniele e Manuele, con i quali condivido un'amicizia sincera e duratura e tutto il gruppo di amici formatosi durante questo percorso triennale con i quali ho trascorso bei momenti.

In ultimo, ma non per importanza, ci terrei a ringraziare gli amici che sono sempre stati al mio fianco sin da prima che questo percorso avesse inizio e sono fieri dei risultati da me raggiunti.

Giacomo Tontini

Indice

Introduzione	III
Definizione degli obiettivi	VII
1 Reperimento e manipolazione dei dati	1
1.1 Fonte dei dati	1
1.1.1 Descrizione degli agenti inquinanti, origine e possibili danni per la salute	3
1.2 Assenza dei dati e relativo completamento	7
1.2.1 Interpolazione lineare	9
1.3 Manipolazione dei dati	11
1.3.1 Unione dei dati meteorologici con quelli sull'inquinamento .	11
1.3.2 Codifica della posizione delle stazioni	11
1.3.3 Codifica delle informazioni temporali	12
1.3.4 Codifica delle informazioni del vento	14
2 Intelligenza artificiale per il forecasting	17
2.1 Che cos'è l'intelligenza artificiale	17
2.1.1 Una prima classificazione	17
2.1.2 Machine Learning e Deep Learning	19
2.2 Reti Neurali Artificiali	20
2.2.1 Tecniche di apprendimento	20
2.2.2 Architettura di una rete neurale	22
2.2.3 Funzione di attivazione	25
2.2.4 Fasi di apprendimento e suddivisione dei dati	28
2.2.5 Overfitting e Underfitting	29
2.3 Configurazione della rete neurale	30
2.3.1 Parametri	30
2.3.2 Ottimizzazione del gradiente	30
2.3.3 Varianti di Gradient descent	32
2.3.4 Iperparametri	34

2.4	Funzioni di errore	38
2.4.1	Mean square error	38
2.4.2	Mean absolute error	38
2.4.3	Huber loss	38
3	Progettazione e Sviluppo	41
3.1	Tecnologie disponibili	41
3.1.1	Tensorflow	41
3.1.2	Keras	42
3.1.3	Numpy e Pandas	42
3.2	Ricerca della funzione di interpolazione ottimale	42
3.3	Funzioni per la valutazione e il confronto dei modelli	44
3.3.1	Funzione di errore	44
3.3.2	Ottimizzatore	44
3.4	Struttura dei dati	44
3.5	Definizione dei modelli	45
3.5.1	Modelli lineari	45
3.5.2	Modelli Gated Recurrent Unit (GRU)	46
3.5.3	Modelli Long Short-Term Memory (LSTM)	48
4	Sperimentazione e Analisi dei risultati	51
4.1	Valutazione dei modelli	51
4.1.1	Modelli lineari	52
4.1.2	Modelli Gated Recurrent Unit (GRU)	54
4.1.3	Modelli Long Short-Term Memory (LSTM)	56
5	Meccanismi avanzati di apprendimento	59
5.1	Modelli stateful	59
5.1.1	Definizione dei modelli	61
5.1.2	Analisi dei risultati	62
5.2	Accenno ai meccanismi Attention	63
6	Ottimizzazione degli iperparametri mediante Grid Search	65
6.1	Approcci di ottimizzazione	65
6.2	Talos	66
6.2.1	Utilizzo del Grid Searching per la scoperta degli iperparametri migliori	67
6.3	Ricerca e analisi dei risultati	68
	Conclusione e sviluppi futuri	IX

Introduzione

Al giorno d'oggi il problema dell'inquinamento sta assumendo un ruolo sempre più importante, sempre più città sparse nel mondo superano più volte all'anno i limiti giornalieri imposti per legge, attuando misure preventive che cercano di limitare i danni, come ad esempio i blocchi sul traffico o il divieto di utilizzo di certe tipologie di riscaldamento domestico. Visto il numero sempre crescente di morti a causa dell'inquinamento [6], in particolare quello derivante dal traffico, risulta necessario tenere costantemente monitorati i valori di inquinamento. Nelle grandi metropoli, in particolare quelle appartenenti a continenti asiatici, capita sempre più spesso di non poter uscire di casa per fare attività fisica o di svago; da questa problematica nasce l'idea di questo progetto di tesi. Il progetto AlmaBike basa le sue fondamenta sul bike sharing, ovvero la possibilità di noleggiare biciclette sparse per la città, utilizzandole all'occorrenza. Grazie alla circolazione continua di questi veicoli, dotati di opportuna sensoristica, sarà possibile effettuare rilevazioni sull'inquinamento in maniera più o meno frequente durante la giornata e nelle aree più frequentate della città di Bologna. L'elevato quantitativo di dati generato, una volta trasmesso ad un sistema di elaborazione centralizzato, potranno essere analizzati e utilizzati per effettuare delle previsioni, consentendo ad esempio, di sapere a priori quale area della città sarà meno inquinata per andare a svolgere attività fisica o quale percorso seguire per giungere a lavoro.

Al fine di realizzare il software che si occupa di effettuare la predizione e considerando che le biciclette non sono ancora in circolazione, sarà necessario in primo luogo cercare una fonte di dati affidabile che consenta di verificare l'effettiva fattibilità di un tale progetto. Una volta ottenuti i dati per effettuare le simulazioni ci si renderà conto di come questi presentino incongruenze e 'buchi temporali' più o meno consistenti, rendendo necessaria una loro manipolazione. Saranno prese in considerazione varie tecniche di interpolazione e i dati codificati mediante one hot encoding per ottenere una base solida da fornire al modulo di predizione.

Grazie al progresso dell'intelligenza artificiale, ovvero la disciplina informatica che consente ad un elaboratore di apprendere, pensare ed agire sui dati in maniera simile alle modalità tipiche del cervello umano, si studieranno delle reti neurali in

grado di carpire le dipendenze presenti tra i vari agenti inquinanti e meteorologici al fine di predirne i valori futuri. Verranno presi in esame numerosi modelli di reti neurali, da quelli più semplici a quelli più complessi, al fine di essere confrontati tra loro. Tali modelli necessitano di essere opportunamente configurati andando ad operare su variabili chiamate 'iperparametri'; in un primo momento si utilizzerà una configurazione frutto di numerose prove manuali svolte in modalità 'prova e sbaglia', successivamente si tratterà uno strumento utile ed efficace che permette l'automazione del processo di ricerca, consentendo di giungere ad una configurazione ottimale.

La comunità scientifica è in continua ricerca di modelli neurali efficienti e concreti, alcuni di questi sono stati particolarmente apprezzati e sfruttati nel tempo. Verranno trattate due modalità operative che modificano il processo di apprendimento dei neuroni e ne alterano il comportamento. La prima basa il suo funzionamento sulla capacità di ricordare informazioni per più o meno tempo. La seconda, similmente al processo di apprendimento dei bambini, è in grado di porre l'attenzione sui dati più interessanti e tralasciare quelli meno rilevanti.

Questo progetto di tesi è stato realizzato insieme a Ruben Ceroni, con il quale sono state effettuate buona parte delle sperimentazioni proposte, nello specifico i primi quattro capitoli sono stati realizzati insieme. Il lavoro svolto è un proseguo di ciò che i colleghi Lorenzo Mondani e Stefano Righini hanno svolto ad inizio 2018.

Il documento sarà strutturato nel modo seguente:

Reperimento e manipolazione dei dati Introduzione al problema, reperimento dei dati e relative manipolazioni.

Intelligenza artificiale per il forecasting Definizione dei concetti base inerenti l'intelligenza artificiale, introduzione alle reti neurali ed i problemi legati al processo di apprendimento. Approfondimento sui parametri ed iperparametri per la configurazione della rete e funzioni principali per la valutazione dell'errore.

Progettazione e Sviluppo Processo di ricerca per il completamento dei dati migliore, strutture utilizzate per la memorizzazione dei dati e definizione dei modelli neurali suddivisi per macro-categorie.

Sperimentazione e Analisi dei risultati Prove per la valutazione dei modelli definiti nel capitolo precedente.

Meccanismi avanzati di apprendimento Definizione dei metodi di apprendimento alterati con relative prove sperimentali e analisi dei risultati

Ottimizzazione degli iperparametri mediante grid search Introduzione alle tipologie di ottimizzazione, definizione del processo di ricerca grid search in riferimento alla libreria utilizzata ed infine prove sperimentali con i risultati dell'ottimizzazione.

Definizione degli obiettivi

In un progetto così ampio è bene individuare a priori quali siano gli obiettivi che si vuole raggiungere ed in particolare definire a quali aspetti dare maggiore importanza.

Utilizzo di informazioni aggiuntive Nel progetto svolto dai colleghi in passato, solamente un sottoinsieme dei dati a disposizione è stato utilizzato (per motivi di performance o di mancanza totale delle informazioni). Il primo obiettivo che ci si pone è quello di poter sfruttare integralmente le informazioni messe a disposizione nei data set di inquinamento dei vari enti di rilevazione. In particolare, un dato ignorato sino ad ora è quello relativo alle coordinate delle stazioni di rilevamento, questo perchè il numero di stazioni utilizzate dai colleghi era di poche unità e di conseguenza poco influente. Pertanto, si ritiene opportuno inserire e manipolare in modo adeguato tali informazioni e nel caso in cui i risultati siano buoni mantenerle nell'insieme.

Scalabilità Uno dei limiti principali nell'utilizzo di sistemi di intelligenza artificiale è l'onerosa quantità di risorse richieste per effettuare le previsioni. Come anticipato il numero di stazioni di rilevazione è stato limitato in passato a poche unità, in particolar modo per evitare di aspettare ore o giorni per dei risultati. L'obiettivo che ci si pone è quello di aumentare in modo significativo il numero di stazioni utilizzate e cercare un modello che consenta di ottenere un buon compromesso tra affidabilità nella predizione e performance nei tempi di apprendimento. Data la volontà di espandere il progetto su scala nazionale, si considera questo requisito di primaria importanza.

Lungimiranza nella predizione Le predizioni effettuate in passato erano sempre inferiori ai quattro giorni di lungimiranza; nonostante questo sia un buon risultato per organizzare la propria giornata sarebbe preferibile riuscire a prevedere i valori di inquinamento per un periodo più lungo, come ad esempio una settimana o una decina di giorni. Si tenga presente come sia naturale aspettarsi che il degrado della predizione aumenti con il passare del tempo in quanto predizioni più future saranno maggiormente influenzate da quelle passate.

Ricerca di una configurazione di rete ottimale Le reti neurali che verranno utilizzate richiedono di essere configurate tramite opportuni parametri per adattare il loro comportamento ai dati e migliorare la predizione. Data la difficoltà riscontrata in passato nell'effettuare questa operazione manualmente, si ritiene necessaria la ricerca di strumenti che semplifichino tale processo e la definizione di un modello di rete neurale che restituisca risultati vicini all'ottimo.

Capitolo 1

Reperimento e manipolazione dei dati

Uno degli obiettivi di questo progetto di tesi consiste nel definire una base solida per la previsione degli agenti inquinanti nelle varie zone della città di Bologna.

Per fare questo, si avranno a disposizione le rilevazioni meteorologiche e degli agenti inquinanti ottenute dalle biciclette dei vari servizi di bike-sharing disponibili in città. In ciascuna bici saranno integrati sensori di inquinamento e di posizione che durante l'uso quotidiano raccoglieranno dati da inviare a un servizio di archiviazione centrale e attraverso l'elaborazione di essi, suggerirà agli utenti il percorso migliore, evitando le zone più inquinate nel momento di percorrenza.

Considerando che il progetto è ancora in fase embrionale, ovvero le bici non dispongono attualmente dei relativi sensori, risulta necessario reperire dati verosimili per realizzare la parte di progetto che realizza la previsione.

1.1 Fonte dei dati

In Italia sono disponibili varie fonti per il reperimento dei dati relativi all'inquinamento atmosferico e a quelli meteorologici; la fonte più autorevole, completa e aggiornata periodicamente è l'archivio gestito dall'Arpa (Agenzia Regionale per la Protezione Ambientale) che gestisce per la sua regione di riferimento varie stazioni di rilevamento, riportate in figura 1.1.

Dato che il progetto AlmaBike è destinato ad essere lanciato nella città di Bologna e l'elevata quantità di stazioni di rilevamento presenti in Emilia-Romagna, la fonte scelta per il reperimento dei dati è stata OpenData, gestito da Arpa-ER [10].

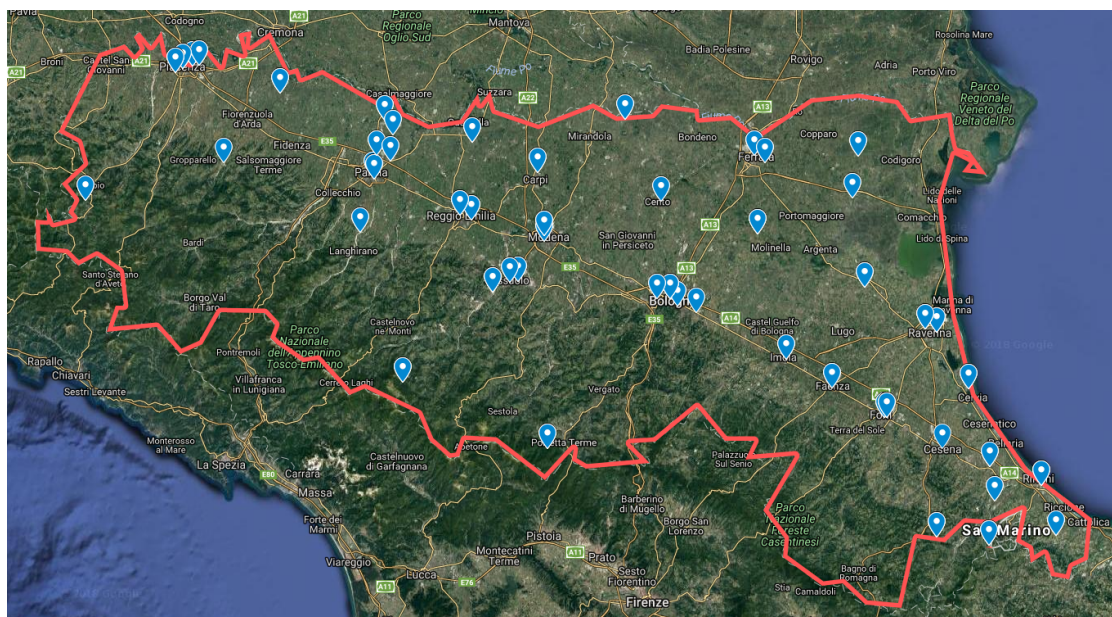


Figura 1.1: Le stazioni Arpae in Emilia-Romagna

OpenData è un archivio suddiviso per tipologia di rilevazione, all'interno del quale sono presenti i file, organizzati anno per anno, con le varie rilevazioni. Sono stati presi in esame gli archivi meteorologici e di inquinamento dell'aria;

Nel dettaglio, gli elementi climatici e relative unità di misura oggetto di rilevazione sono riportati in tabella 1.1 e gli agenti inquinanti in tabella 1.2.

Tabella 1.1: Elementi climatici e frequenza di rilevazione

<i>Tipo di dato</i>	<i>Unità di misura</i>	<i>Frequenza</i>
Temperatura	K	Oraria
Pressione	Pa	Oraria
Umidità	g/m^3	Oraria
Velocità del vento	km/h	Oraria
Direzione del vento	$0^\circ - 360^\circ$	Oraria
Pioggia	mm	Oraria

Tabella 1.2: Agenti inquinanti e frequenza di rilevazione

<i>Tipo di dato</i>	<i>Descrizione</i>	<i>Unità di misura</i>	<i>Frequenza</i>
PM10	Particulate Matter	K	Giornaliera
PM2.5	Particulate Matter	Pa	Giornaliera
O ₃	Ozono	g/m ³	Oraria
NO ₂	Biossido di azoto	km/h	Oraria
C ₆ H ₆	Benzene	0°—360°	Giornaliera
CO	Monossido di carbonio	µg m ⁻¹	Oraria
SO ₂	Biossido di zolfo	µg m ⁻¹	Oraria
NOX	Ossidi di azoto	µg m ⁻¹	Oraria
C ₆ H ₅ —CH ₃	Toluene	µg m ⁻¹	Giornaliera
NO	Monossido di azoto	µg m ⁻¹	Oraria
C ₈ H ₁₀	o-xylene	µg m ⁻¹	Giornaliera

1.1.1 Descrizione degli agenti inquinanti, origine e possibili danni per la salute

Particolato PM10

Il particolato è l'inquinante atmosferico che provoca i maggiori danni alla salute umana in Europa ed è costituito da particelle così leggere che possono fluttuare nell'aria. Si tratta di particelle solide e liquide aventi diametro variabile fra 0,1 e circa 100 µm.

Il termine PM10 identifica le particelle di diametro aerodinamico inferiore o uguale ai 10 µm (1 µm). Le particelle PM10 penetrano in profondità nei nostri polmoni. Il loro effetto sulla nostra salute e sull'ambiente dipende dalla loro composizione e a seconda di questa, le particelle possono anche avere effetti sul clima globale, sia riscaldando che raffreddando il pianeta.



Figura 1.2: Limiti di legge per il PM 10

Alcune particelle vengono emesse direttamente nell'atmosfera, altre si formano come risultato di reazioni chimiche che coinvolgono i gas precursori, vale a dire l'anidride solforosa, gli ossidi di azoto, l'ammoniaca e i composti organici volatili. Gran parte delle particelle emesse direttamente derivano dalle attività umane, principalmente dalla combustione di combustibili fossili e biomasse. Un importante contributo alle emissioni di particelle è rappresentato dai gas di scarico dei veicoli con motori a combustione interna, ma anche dall'usura degli pneumatici, dei freni e dell'asfalto.

L'origine dell'inquinamento da PM10 varia sensibilmente da zona a zona e nel corso del tempo. Si stima che, in media, in Emilia-Romagna la parte preponderante dell'inquinamento da PM10 sia dovuto alle attività umane, con una frazione variabile tra il 75% in Appennino e l'85% in pianura.

Particolato PM2.5

L'inquinamento da particolato fine (PM2.5, ossia particolato con un diametro minore di $2,5\ \mu$) è composto da particelle solide e liquide così piccole che non solo penetrano in profondità nei nostri polmoni, ma entrano anche nel nostro flusso sanguigno, proprio come l'ossigeno.



Figura 1.3: Limiti di legge per il PM 2.5

Un recente studio dell'Organizzazione mondiale della sanità dimostra che l'inquinamento da particolato fine potrebbe essere un problema per la salute maggiore di quanto si pensasse in precedenza. Secondo il rapporto dell'OMS «Rassegna delle prove sugli aspetti sanitari dell'inquinamento atmosferico», un'esposizione prolungata al particolato fine può scatenare l'arteriosclerosi, creare problemi alla nascita e malattie respiratorie nei bambini. Lo studio inoltre suggerisce un possibile collegamento con lo sviluppo neurologico, le funzioni cognitive e il diabete, e rafforza il nesso di causalità tra PM2,5 e morti cardiovascolari e respiratorie.

Alcuni componenti del particolato fine vengono emesse direttamente nell'atmosfera, altri si formano come risultato di reazioni chimiche che coinvolgono i gas precursori. Il nerofumo, uno dei componenti comuni della fuliggine rilevato

principalmente nel particolato fine, è il risultato della combustione incompleta di combustibili, sia di combustibili fossili che del legno. Nelle aree urbane le emissioni di nerofumo sono causate principalmente dal trasporto stradale, in particolare dai motori diesel.

Ozono

L'ozono (O_3) è una forma speciale e altamente reattiva di ossigeno ed è composto da tre atomi di ossigeno. Nella stratosfera, uno degli strati più alti dell'atmosfera, l'ozono ci protegge dalle pericolose radiazioni ultraviolette provenienti dal sole.

Nello strato più basso dell'atmosfera – la troposfera – l'ozono è, di fatto, un'importante sostanza inquinante che influisce sulla salute pubblica e l'ambiente. Alti livelli di ozono corrodono i materiali, gli edifici e i tessuti vivi, inoltre, riduce la capacità delle piante di eseguire la fotosintesi e ostacola il loro assorbimento di anidride carbonica, indebolendone la crescita e la riproduzione, con il risultato di minori raccolti e di uno sviluppo ridotto di boschi e foreste. Nel corpo umano provoca infiammazioni ai polmoni e ai bronchi. Non appena esposto all'ozono, il nostro corpo cerca di impedirne l'entrata nei polmoni. Questa reazione riduce l'ammontare di ossigeno che inaliamo. Inalare meno ossigeno rende il lavoro del cuore più difficile. Quindi per le persone che già soffrono di disturbi cardiovascolari o respiratori, come l'asma, picchi di ozono possono essere debilitanti e persino fatali.

Le reazioni chimiche che producono ozono sono catalizzate dalla radiazione solare, di conseguenza questo inquinante è tipicamente estivo e assume valori di concentrazione più elevati nelle estati contrassegnate da alte temperature e elevata insolazione. L'immissione di inquinanti primari (prodotti dal traffico, dai processi di combustione, dai solventi delle vernici, dall'evaporazione di carburanti, etc.) favorisce quindi la produzione di un eccesso di ozono rispetto alle quantità altrimenti presenti in natura durante i mesi estivi.

Biossido d'azoto

Il biossido di azoto (NO_2) è un gas reattivo, di colore bruno e di odore acre e pungente. L' NO_2 è un importante inquinante dell'aria che, come l'ozono, risulta dannoso per il sistema respiratorio.

L'esposizione a breve termine all' NO_2 può causare diminuzione della funzionalità polmonare, specie nei gruppi più sensibili della popolazione, mentre l'esposizione a lungo termine può causare effetti più gravi come un aumento della suscettibilità alle infezioni respiratorie. L' NO_2 è fortemente correlato con altri inquinanti, come il PM, perciò negli studi epidemiologici è difficile differenziarne gli effetti dagli altri

inquinanti. Gli ossidi di azoto giocano un ruolo principale nella formazione di ozono e contribuiscono alla formazione di aerosol organico secondario, determinando un aumento della concentrazione di PM10 e PM2,5.



Figura 1.4: Limiti di legge per il NO₂

Il biossido di azoto (NO₂) si forma prevalentemente dall'ossidazione di monossido di azoto (NO). Questi due gas sono noti con il nome di NO_x. Le maggiori sorgenti di NO ed NO₂ sono i processi di combustione ad alta temperatura (come quelli che avvengono nei motori delle automobili o nelle centrali termoelettriche). L'NO rappresenta la maggior parte degli NO_x emessi; per gran parte delle sorgenti, solo una piccola parte di NO_x è emessa direttamente sotto forma di NO₂ (tipicamente il 5-10%). Fanno eccezione i veicoli diesel, che emettono una proporzione maggiore di NO₂, fino al 70% degli NO_x complessivi, a causa del sistema di trattamento dei gas di scarico di questi veicoli.

Benzene

Il benzene (C₆H₆) è una sostanza chimica liquida e incolore dal caratteristico odore aromatico pungente. A temperatura ambiente volatilizza assai facilmente, cioè passa dalla fase liquida a quella gassosa.

L'effetto più noto dell'esposizione cronica riguarda la potenziale cancerogenicità del benzene sul sistema emopoietico (cioè sul sangue). L'Agenzia Internazionale per la Ricerca sul Cancro (IARC) classifica il benzene come sostanza cancerogena di classe I, in grado di produrre varie forme di leucemia.

La maggior parte del benzene oggi prodotto (85%) trova impiego nella chimica come materia prima per numerosi composti secondari, a loro volta utilizzati per produrre plastiche, resine, detergenti, pesticidi, intermedi per l'industria farmaceutica, vernici, collanti, inchiostri, adesivi e prodotti per la pulizia. Il benzene è inoltre contenuto nelle benzine in cui viene aggiunto, insieme ad altri composti aromatici, per conferire le volute proprietà antidetonanti e per aumentarne il "nu-

mero di ottani”, in sostituzione totale (benzina verde) o parziale (benzina super) dei composti del piombo.

Monossido di carbonio

Il monossido di carbonio (CO), incolore e inodore, è un tipico prodotto derivante dalla combustione. Il CO viene formato in modo consistente durante la combustione di combustibili con difetto di aria e cioè quando il quantitativo di ossigeno non è sufficiente per ossidare completamente le sostanze organiche.

A bassissime dosi il CO non è pericoloso, ma già a livelli di concentrazione nel sangue pari al 10-20% il soggetto avverte i primi sintomi dovuti all'esposizione a monossido di carbonio, quali lieve emicrania e stanchezza. La principale sorgente di CO è storicamente rappresentata dal traffico veicolare (circa l'80% delle emissioni a livello mondiale), in particolare dai gas di scarico dei veicoli a benzina. La concentrazione di CO emessa dagli scarichi dei veicoli è strettamente connessa alle condizioni di funzionamento del motore: si registrano concentrazioni più elevate con motore al minimo e in fase di decelerazione, condizioni tipiche di traffico urbano intenso e rallentato.

Biossido di zolfo

Gli ossidi di zolfo sono caratterizzati dall'assenza di colore, l'odore acre e pungente e l'elevata reattività a contatto con l'acqua e sono genericamente indicati come SOx. A livello antropico, SO₂ e SO₃ sono prodotti nelle reazioni di ossidazione per la combustione di materiali in cui sia presente zolfo quale contaminante [9].

1.2 Assenza dei dati e relativo completamento

A seguito del download dei file dall'archivio e dell'importazione in apposite strutture in un database temporaneo è stato possibile notare come molti degli agenti inquinanti fossero in realtà assenti per periodi di tempo notevoli (nell'ordine dei mesi) e nei periodi in cui le rilevazioni sono presenti la loro periodicità è poco frequente (una volta al giorno o addirittura una volta a settimana).

Considerando che il progetto di AlmaBike si pone come obiettivo quello di effettuare delle previsioni sull'andamento degli inquinanti ora per ora, si è scelto di utilizzare fra gli agenti precedentemente presi in esame il Biossido di Azoto che, come mostrato dalla tabella 1.3 è quello misurato più consistentemente e con la minor quantità di valori mancanti.

Nonostante le rilevazioni di questo inquinante negli archivi siano le più complete e frequenti, sono comunque presenti tantissimi record mancanti che creano non pochi problemi nelle reti neurali che verranno descritte nei prossimi capitoli. A

Tabella 1.3: Riassunto dei valori mancanti

<i>Stazione</i>	<i>Parametro</i>	<i>Valori mancanti</i>	<i>% mancante</i>
2000003	NO ₂	13002	17.6
	PM10	70749	95.9
	PM2.5	71138	96.5
	O ₃	4569	6.2
2000004	PM10	70769	96.0
	C ₆ H ₆	6031	8.2
	CO	5184	7.0
	NO	5181	7.0
	NO ₂	5181	7.0
	NOX	5181	7.0
2000219	PM10	71854	97.4
	NO ₂	5153	7.0
	O ₃	30496	41.3
	NO	5153	7.0
	PM2.5	71949	97.6
	NOX	5151	7.0
2000229	PM10	71963	97.6
	PM2.5	71924	97.5
	NO ₂	31492	42.7
2000232	C ₆ H ₆	33767	45.8
	NO ₂	31478	42.7
	PM2.5	71916	97.5
	PM10	71924	97.5
4000152	PM10	72139	97.8
	O ₃	10088	13.7
	PM2.5	71471	96.9
	NO ₂	2242	3.0
4000155	PM10	70817	96.0
	O ₃	34229	46.4
	NO ₂	2083	2.8
	NO	1931	2.6
	PM2.5	72498	98.3
	NOX	23025	31.2

tale scopo si è scelto di sfruttare tecniche di interpolazione lineare per raggiungere il completamento del 100% della base di dati.

1.2.1 Interpolazione lineare

In matematica, per interpolazione si intende un metodo per individuare nuovi punti del piano cartesiano a partire da un insieme finito di punti dati, nell'ipotesi che tutti i punti si possano riferire ad una funzione $f(x)$ di una data famiglia di funzioni di una variabile reale.

Sia data una sequenza di n numeri reali distinti x_k chiamati nodi e per ciascuno di questi x_k sia dato un secondo numero y_k . Si ponga come obiettivo quello di individuare una funzione f di una certa famiglia tale che sia $f(x_k)=y_k$ per $k=1,2,\dots,n$

Tabella 1.4: Dati esempio interpolazione

x	$f(x)$
0	0
1	0.8415
2	0.9093
3	0.1411
4	-0.7568
5	-0.9589
6	-0.2794

Una coppia (x_k, y_k) viene chiamato punto dato ed f viene detta funzione interpolante, o semplicemente interpolante, per i punti dati. L'interpolazione risolve problemi come: "quanto vale la funzione in $x=2.5$?"

Esistono molti metodi differenti di interpolazione, l'interpolazione lineare, polinomiale e razionale: ovvero l'interpolazione mediante funzioni razionali e l'interpolazione trigonometrica che si serve di polinomi trigonometrici. Per questioni di efficienza e di semplicità a livello implementativo si è scelto di utilizzare l'interpolazione lineare attraverso la libreria Numpy, che verrà descritta nei capitoli implementativi.

Si consideri il suddetto esempio di determinare $f(2.5)$. Poiché 2.5 è il punto medio fra 2 e 3, è ragionevole assegnare a $f(2.5)$ come il valore medio fra $f(2)=0.9093$ e $f(3)=0.1411$ in tal modo si ottiene $f(2.5) = 0.5252$. In generale, l'interpolazione lineare per ogni coppia di punti dati consecutivi, (x_a, y_a) e (x_b, y_b) , definisce come funzione interpolante nell'intervallo $[x_a, x_b]$ la

$$f(x) = \frac{x - x_b}{x_a - x_b} y_a - \frac{x - x_a}{x_a - x_b} y_b$$

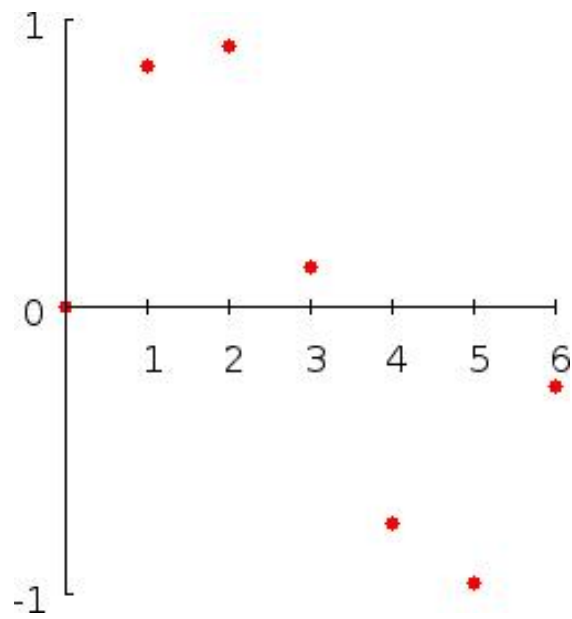


Figura 1.5: Punti del grafo ai quali si vuole applicare l'interpolazione

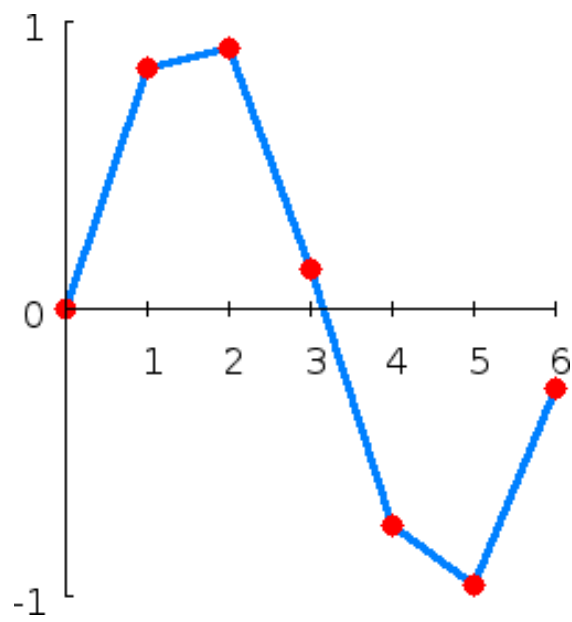


Figura 1.6: Esempio di riempimento mediante interpolazione lineare

Questa formula può essere interpretata come valutazione della media ponderata [22].

1.3 Manipolazione dei dati

A seguito del riempimento dei dati mancanti mediante interpolazione lineare si è reso necessario effettuare svariate operazioni per poter passare alla rete neurale i dati; Le operazioni più importanti effettuate e non standard (ovvero dipendenti dalla tipologia dei dati del problema) sono:

- Unione dei due set di dati, ovvero quello relativo ai dati meteorologici e quello relativo all'inquinamento;
- Associazione della posizione alle stazioni e aggiunta al dataset;
- Codifica delle informazioni sulla data e l'ora dei record di misurazione e aggiunta al dataset
- Codifica vettoriale o tramite OneHotEncoding delle informazioni del vento

1.3.1 Unione dei dati meteorologici con quelli sull'inquinamento

Le varie stazioni Arpae presenti sul territorio nazionale effettuano periodicamente le misurazioni, ogni stazione rileva solamente un sottoinsieme degli agenti inquinanti o degli agenti atmosferici presentati precedentemente; questo costituisce un problema di natura prettamente implementativa in quanto ogni stazione deve avere necessariamente gli stessi parametri misurati con la stessa frequenza. Un semplice quanto efficace metodo per risolvere questa problematica consiste nell'associare per ogni stazione di rilevamento meteo la stazione degli agenti inquinanti più vicina ad essa (geograficamente) ed eliminare eventuali agenti non presenti in tutto l'insieme delle stazioni risultanti da questa operazione di unione. Fortunatamente tutte le stazioni meteorologiche effettuano le misurazione degli stessi agenti atmosferici, mentre per quanto riguarda gli agenti inquinanti, per i motivi definiti precedentemente, si considerano solamente il Biossido d'azoto NO_2 . Delle 200 e oltre stazioni di rilevamento presenti sul territorio emiliano romagnolo, in seguito a tale operazione si è ottenuto un set di 42 stazioni.

1.3.2 Codifica della posizione delle stazioni

Le varie stazioni di rilevazione sono sparse lungo tutta l'Emilia-Romagna e per ogni provincia si hanno mediamente 3 stazioni di rilevamento. Con queste

condizioni i dati per singola città non sono sufficienti per effettuare previsioni sul futuro e soprattutto atte a simulare la quantità di dati che le future biciclette in giro per Bologna genereranno. Un ottimo workaround a questa problematica è stato quello di considerare tutte le stazioni della regione come appartenenti al centro di Bologna e organizzate sotto forma di griglia (vedi figura:1.7).



Figura 1.7: Rappresentazione logica delle stazioni a Bologna

A titolo esemplificativo, se una stazione precedentemente aveva come coordinate 50° latitudine e 70° longitudine, una volta traslate diventeranno [5,7]. E' importante notare come, data la conformazione della città e il numero di stazioni (42) si sia cercato di coprire nel miglior modo possibile l'intera città adottando una griglia 6x7 e quindi somigliante il più possibile ad un quadrato. Se il numero delle stazioni fosse stato maggiore e la città avesse avuto una conformazione differente (ad esempio simile ad un rettangolo) sarebbe stato necessario modificare tale configurazione aumentandone la densità e forma della griglia.

1.3.3 Codifica delle informazioni temporali

Variabili Categoricali

Una variabile categorica è una variabile che contiene valori di etichetta invece di un dato numerico, spesso in un range limitato. Per esempio una variabile

categorica può essere "libro", che assume valori "fantasy", "romanzo" e "giallo" oppure la variabile "posto" con valori "primo", "secondo" e "terzo". Alcune categorie possono avere una relazione naturale tra di loro, come la variabile "posto" che ha un ordinamento naturale dei valori.

Il problema nell'uso dei dati categorici è dato dal fatto che la maggior parte dei modelli si aspetta in input valori numerici invece che etichette e ciò richiede la trasformazione di esse in valori numerici, effettuata generalmente in due step: Integer encoding e One-Hot encoding.

Integer encoding

Consiste nell'assegnare a ogni possibile valore di una variabile categorica un numero intero, facendo così nel caso di "libro" avremmo: "fantasy" \rightarrow 1, "romanzo" \rightarrow 2, "giallo" \rightarrow 3. Nel caso di variabili ordinali come "posto" può essere sufficiente questo step dato che l'algoritmo utilizzato potrebbe essere in grado di comprendere la relazione che intercorre tra i valori.

One-Hot encoding

Nel caso di variabili categoriche tra le quali non esiste alcuna relazione ordinale viene utilizzata la tecnica del one-hot encoding, in quanto con il solo integer encoding si introdurrebbero relazioni di ordine naturale tra le categorie, influenzando negativamente i risultati. Tale tecnica consiste nel trasformare ogni variabile integer-encoded in più variabili binarie, tante quanti sono i possibili valori differenti della codifica precedente, impostando il valore "1" in corrispondenza del valore assunto dalla variabile e "0" per gli altri valori [14].

One-Hot Encoding del tempo

Si è deciso di effettuare un codifica One-hot sulle informazioni temporali, soprattutto per introdurre un'informazione di circolarità e per rappresentare situazioni che si ripetono temporalmente, come il traffico di punta in certe ore e in certi giorni della settimana, oppure gli esodi estivi ed invernali, ragion per cui la codifica One-hot è stata fatta sui giorni della settimana, sui mesi e sulle ore.

Tabella 1.5: Dato temporale originale

<i>DateTime</i>
2017-12-30 23:00:00
2017-12-31 23:00:00

Tabella 1.6: Dato temporale codificato ad interi

<i>Day</i>
6
7

Tabella 1.7: Dato temporale codificato One-Hot

<i>D_1</i>	<i>D_2</i>	<i>D_3</i>	<i>D_4</i>	<i>D_5</i>	<i>D_6</i>	<i>D_7</i>
0	0	0	0	0	1	0
0	0	0	0	0	0	1

1.3.4 Codifica delle informazioni del vento

Il vento è un agente atmosferico molto importante nell'ambito della meteorologia in quanto determina in modo significativo la variazione della concentrazione di tutti gli altri agenti. E' opportuno pensare come le informazioni sulla velocità e sulla direzione del vento debbano essere opportunamente manipolate per esprimere nel modo più semplice ma significativo possibile la loro valenza espressiva. Di seguito verranno illustrate due differenti modalità con le quali tali informazioni sono state modificate, sono entrambe molto valide e largamente utilizzate. E' importante sottolineare come queste siano mutualmente esclusive.

One-Hot Encoding del vento

Analogamente al tempo, anche per la direzione del vento può essere sfruttata la codifica One-Hot, tuttavia si rende necessario effettuare una prima semplificazione per evitare il proliferarsi di 360 valori distinti. Tenendo in considerazione il range di valori che la direzione del vento può assumere (da 0° a 360°) si è deciso di rapportare l'intervallo nelle 8 direzioni cardinali più comuni: Est, Nord-Est, Nord, Nord-Ovest, Ovest, Sud-Ovest, Sud, Sud-Est. Di seguito la funzione utilizzata che sfrutta l'operazione di modulo e divisione:

$$f(angle) = angle \bmod 360 : 45$$

In questo modo, ogni possibile angolo viene approssimato a quello più vicino tra: 0°, 45°, 90°, 135°, 180°, 225°, 270°, 315°. Eseguita tale operazione, l'operazione di One-hot encoding della direzione risulta analoga a quella del tempo. Naturalmente la velocità del vento, con questa codifica, rimane invariata.

Codifica vettoriale del vento

Una codifica più interessante ed espressiva è la codifica delle informazioni sul vento in termini di vettori. Sfruttando i vettori è possibile combinare l'informazione della velocità con quella della direzione in una nuova informazione sicuramente più rappresentativa in termini numerici.

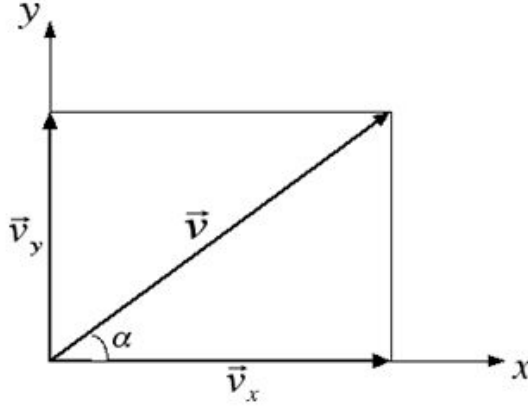


Figura 1.8: Rappresentazione di un generico vettore

In generale, posto un vettore \vec{V} nel piano e scelto un opportuno sistema di riferimento cartesiano, si possono individuare le due componenti x e y del vettore; si indicheranno con \vec{V}_x e \vec{V}_y le due proiezioni del segmento orientato lungo i due assi [16]. Come è possibile notare in figura 1.8, il vettore risulta inclinato di un angolo α rispetto all'asse orizzontale e per questo motivo i moduli delle due componenti x e y risultano per ragioni di trigonometria:

$$|\vec{V}_x| = |\vec{V}| \cdot \cos \alpha$$

$$|\vec{V}_y| = |\vec{V}| \cdot \sin \alpha$$

Nello specifico la direzione del vento definisce l'angolo del vettore, mentre la velocità ne costituisce la lunghezza. In tal modo abbiamo modificato due informazioni che per una persona umana sono intuitivamente correlate fra loro, mentre per una macchina che capisce solo numeri e basa le sue intuizioni su questi ultimi non lo è altrettanto.

Arrivati alla fine di questo capitolo i dati a disposizione sono pronti per essere dati in pasto alla rete neurale, le cui fondamenta verranno descritte nel prossimo capitolo.

Capitolo 2

Intelligenza artificiale per il forecasting

In questo capitolo verrà descritta in maniera approfondita che cos'è l'intelligenza artificiale e le reti neurali che verranno sfruttate per effettuare delle previsioni sugli agenti inquinanti nella città di Bologna. Verranno descritte le fondamentali principali di una rete neurale ed in particolare i parametri e iperparametri che ne definiscono il comportamento.

2.1 Che cos'è l'intelligenza artificiale

L'intelligenza artificiale è l'abilità di un computer ad acquisire e rappresentare la conoscenza, apprendere e sviluppare una capacità decisionale autonoma che gli consenta di risolvere i problemi e affrontare i fenomeni di un ambiente operativo complesso. In modo semplicistico potremmo definire l'intelligenza artificiale come l'abilità di un sistema tecnologico di risolvere problemi o svolgere compiti e attività tipiche della mente e delle abilità umane. Guardando al settore informatico, potremmo identificare l'AI – Artificial Intelligence - come la disciplina che si occupa di realizzare macchine (hardware e software) in grado di "agire" autonomamente al fine di risolvere problemi e decidere come e quali azioni intraprendere. Dare una definizione esatta di Intelligenza Artificiale è un compito arduo ma, osservandone le evoluzioni storiche, si è in grado di tracciarne i contorni e quindi di fare alcune importanti classificazioni.

2.1.1 Una prima classificazione

Prendendo come base di partenza il funzionamento del cervello umano e pur sapendo che ancora oggi non se ne comprende ancora a fondo l'esatto funziona-

mento, l'Intelligenza Artificiale dovrebbe saper emulare alcune azioni tipiche del cervello umano:

- agire umanamente in modo indistinto rispetto ad un essere umano;
- pensare umanamente risolvendo un problema con funzioni cognitive;
- pensare razionalmente sfruttando cioè la logica di un essere umano;
- agire razionalmente avviando un processo per ottenere il miglior risultato atteso in base alle informazioni a disposizione, che è ciò che un essere umano, spesso anche inconsciamente, compie abitualmente.

Queste considerazioni sono di assoluta importanza perché permettono di classificare l'AI in due grandi "filoni" di indagine, ricerca e sviluppo in cui la comunità scientifica si è trovata concorde, quello dell'AI debole e dell'AI forte:

Intelligenza Artificiale debole (weak AI): Identifica sistemi tecnologici in grado di simulare alcune funzionalità cognitive dell'uomo senza però raggiungere le reali capacità intellettuali tipiche di quest'ultimo, parliamo di programmi matematici di problem-solving con cui si sviluppano funzionalità per la risoluzione dei problemi o per consentire alle macchine di prendere decisioni.

Intelligenza Artificiale forte (strong AI): In questo caso si parla di "sistemi sapienti" (alcuni scienziati si spingono a definirli addirittura "coscienti di sé") che possono quindi sviluppare una propria intelligenza senza emulare processi di pensiero o capacità cognitive simili all'uomo, sviluppandone una propria in modo autonomo.

Fino alla fine degli anni '80, gli studiosi hanno creduto fortemente nell'AI forte. Un esempio su tutti sono i robot di Isaac Asimov, scrittore che ha consegnato ai posteri un'opera immane dove i protagonisti sono dei robot che ragionano come un essere umano e in determinate situazioni in modo superiore ad esso. I funzionalisti, ovvero gli studiosi che più hanno creduto nell'intelligenza artificiale, sostengono quindi che è possibile creare una macchina artificiale che sia, in tutto e per tutto, uguale e superiore alla mente umana. E come si fa a stabilire se un sistema artificiale sia o meno intelligente? Tramite il test di Turing. Se una macchina supera tale test, allora può scientificamente definirsi intelligente. Tale test è semplicissimo: Un uomo chiuso in una stanza pone delle domande attraverso una tastiera remota ad un calcolatore. Se l'uomo non riesce a capire chi dall'altra parte della stanza gli fornisca le risposte, allora siamo in presenza di un calcolatore intelligente. Anche se nessuna macchina ha mai superato con successo tale test, i fervidi sostenitori dell'intelligenza artificiale forte affermano che è semplicemente una questione di tempo. È forse la teoria più affascinante e contemporaneamente quella che da un punto di vista filosofico e morale pone più interrogativi. Potrà mai

una macchina produrre opere d'arte migliori di quelle umane o risolvere questioni filosofiche?.

I sostenitori della concezione debole, come Hubert Dreyfus, affermarono un concetto semplicissimo: l'intelligenza artificiale non è una vera e propria intelligenza, come faceva pensare invece il test di Turing. Dreyfus, in particolare, contestò nel corso degli anni il concetto di AI forte. Il concetto dal quale si parte è completamente errato. Un calcolatore utilizza dati attraverso delle regole ben precise, ma è capace quindi di cogliere solo alcuni aspetti della realtà, al contrario della mente che ragiona cogliendo i vari elementi nella loro totalità. John Searle ideò un test, denominato "test della stanza cinese" da contrapporre al famoso test di Turing. Lo studioso dimostrò che una macchina, pur avendo superato con successo il primo test, non può essere definita intelligente. Questo perché il calcolatore ha svolto un'azione su una serie di dati che gli sono stati consegnati. La macchina, grazie a un opportuno programma, manipola le parole ma non ne comprende assolutamente il senso. La finalità di Searle era chiara: senza comprensione del significato del linguaggio una macchina non potrà mai essere definita intelligente. Quindi in un'ipotetica disputa tra forte e debole, qual è l'Intelligenza vincente? Le moderne teorie hanno superato questa contrapposizione offrendo una nuova risposta a tale domanda. Una macchina potrà essere definita intelligente solo quando potrà riprodurre il funzionamento del cervello a livello cellulare [20].

2.1.2 Machine Learning e Deep Learning

La classificazione AI debole e AI forte sta alla base della distinzione tra Machine Learning e Deep Learning, due ambiti di studio che rientrano nella più ampia disciplina dell'intelligenza Artificiale che meritano di essere contraddistinti. Ciò che caratterizza l'Intelligenza Artificiale da un punto di vista tecnologico e metodologico è il metodo di apprendimento, ovvero il modo con cui l'intelligenza diventa abile in un compito o azione. Questi metodi di apprendimento sono ciò che distinguono Machine Learning e Deep Learning e sono in gergo chiamati "modelli".

Machine Learning

Si tratta di un'insieme di metodi per consentire al software di adattarsi, metodi attraverso i quali si permette alle macchine di apprendere in modo che possano poi svolgere un compito o una attività senza che siano preventivamente programmati, ovvero senza che vi sia un sistema che stabilisca come deve comportarsi e reagire. In altre parole, si tratta di sistemi che servono ad "allenare" l'AI affinché imparando, correggendo gli errori e allenando sé stessa possa poi svolgere autonomamente un determinato compito od attività.

Deep Learning

In questo caso parliamo di modelli di apprendimento ispirati alla struttura ed al funzionamento del cervello biologico e di conseguenza della mente umana. Se il Machine Learning può essere definito come il metodo che “allena” l’AI, il Deep Learning è quello che permette di emulare la mente dell’uomo. In questo caso, però, il modello matematico da solo non basta, il Deep Learning necessita di reti neurali artificiali progettate ad hoc (deep artificial neural networks) e di una capacità computazionale molto potente capace di “reggere” differenti strati di calcolo e analisi (che è quello che succede con le connessioni neurali del cervello umano) [11].

Quest’ultima tipologia sarà quella che concretamente verrà impiegata nel corso di questo progetto in quanto la difficoltà del problema e la quantità di dati a disposizione necessitano di modelli atti a comprendere ed in particolare apprendere nel modo più approfondito possibile le relazioni fra le varie tipologie di dato tenendo in considerazione anche lo scorrere del tempo. Una tipologia specifica facente parte della branchia di applicazioni del Deep Learning si identifica in quelle che sono chiamate reti neurali, ovvero dei modelli matematici composti da neuroni "artificiali".

2.2 Reti Neurali Artificiali

Una rete neurale artificiale (Artificial Neural Network ANN), in gergo chiamata rete neurale è un sistema dinamico avente la topologia di un grafo orientato avente come nodi, i neuroni artificiali ed archi, i pesi sinaptici. Il termine rete è riferito alla topologia dei collegamenti tra i neuroni. La rete può essere vista come una scatola nera di cui si può ignorare il funzionamento, che associa un input a un output. A livello scientifico non è altro che un modello matematico che applica una funzione $f(input, pesi)$ il cui comportamento varia in funzione dei pesi e naturalmente dell’input, senza specificare la forma della funzione f .

2.2.1 Tecniche di apprendimento

Nell’ambito dell’intelligenza artificiale vi sono svariate tecniche di apprendimento, ovvero metodologie che consentono al sistema di imparare dai dati forniti, per semplicità di trattazione verranno descritte le tecniche più note in letteratura dalle quali sono state derivate le altre nel corso degli anni.

Apprendimento supervisionato

L'Apprendimento supervisionato è una tecnica di apprendimento che mira a istruire un sistema informatico in modo da consentirgli di risolvere dei compiti in automatico. In questa modalità si fornisce alla rete un insieme di dati in ingresso I (tipicamente in forma vettoriale) e un insieme di dati in uscita O . A seconda che questi ultimi siano dei valori continui o meno ci si ritrova di fronte ad un problema di regressione o classificazione (anche comunemente chiamata etichettatura). Si definisce supervisionato in quanto oltre ai dati viene definita anche la funzione $f(x)$ che associa ai dati di ingresso I_x i dati in uscita U_k .

Tutti gli algoritmi di apprendimento supervisionato partono dal presupposto che se forniamo ad esso un numero adeguato di esempi questo sarà in grado di creare una funzione h_1 che approssimerà la funzione h . Fornendo dei dati in ingresso mai analizzati la funzione dovrebbe essere in grado di fornire delle risposte in uscita simili a quelle fornite da h e quindi corrette. Nella realtà questo comportamento non si verifica mai esattamente come descritto, basti pensare alle dinamiche caotiche legate al tempo. Si può facilmente intuire che il buon funzionamento di questi algoritmi dipende in modo significativo dai dati in ingresso (vedi 2.2.5).

Esempio: se un medico ha interesse nel sapere se un paziente è malato o meno di una determinata malattia, alla macchina vengono consegnate una serie di casistiche precedenti, dove l'esito malato/sano è già stato definito. Sulla base di tale casistiche, si allena il sistema a riconoscere appunto l'esito in nuovi casi non ancora capitati.

Apprendimento non supervisionato

Nel caso di apprendimento non supervisionato, la logica di classificazione non viene consegnata alla macchina, vengono forniti solo i dati di input e sarà l'algoritmo ad assegnare la classe di appartenenza. Se ad esempio si ha una popolazione da classificare in gruppi, a seconda di caratteristiche definite, si lascia al programma la definizione dei raggruppamenti.

Reinforcement learning

Il reinforcement learning è una tecnica di apprendimento che basa il suo funzionamento sull'interazione con l'ambiente circostante nel quale l'agente dotato di intelligenza artificiale viene fatto operare. Si differenzia con gli altri approcci di apprendimento automatico in quanto all'algoritmo non viene esplicitamente spiegato come eseguire un'attività. L'agente, che potrebbe essere un'auto a guida autonoma o un programma che gioca a scacchi, interagisce con il suo ambiente, ricevendo una ricompensa ("reward") in base alle sue azioni: ad esempio se arriva

a destinazione in sicurezza o vince una partita. Altrimenti, l'agente riceve una penalità per l'esecuzione scorretta, come andare fuori strada o essere sotto scacco matto. L'agente possiede come unico obiettivo iniziale quello di prendere le decisioni che massimizzino la propria ricompensa.

2.2.2 Architettura di una rete neurale

Fino ad ora si è genericamente parlato di una rete neurale in termini di nodi e archi che insieme vanno a costituire un grafo connesso al fine di realizzare una struttura che assomigli il più possibile alla rete di neuroni del cervello umano. In realtà, una rete neurale è una struttura incredibilmente più semplice di un cervello: i neuroni di una rete neurale sono soltanto dei numeri, il cui valore può andare da 0 a 1. Vengono chiamati neuroni non perché sono simili ai singoli neuroni del cervello, ma per via del modo in cui sono organizzati: essi sono infatti raggruppati in layers, o livelli.

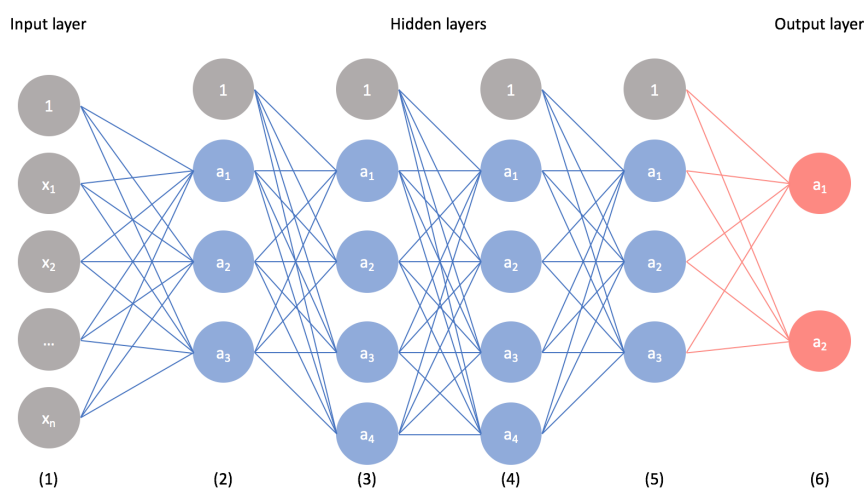


Figura 2.1: Architettura di una rete neurale con hidden layer [21]

I neuroni del primo livello sono detti neuroni di input, sono quelli incaricati di tradurre il dato che viene fornito come input, come ad esempio una foto o un testo, in un dato che la rete neurale può comprendere. Nel caso di un'immagine, ogni neurone di input corrisponderà a un pixel, e ci sarà un numero che rappresenta il colore di quel pixel. I neuroni dell'ultimo livello sono detti neuroni di output e in questo caso ogni neurone corrisponde a una delle possibili risposte che la rete neurale può dare. Ad esempio, se la nostra rete deve riconoscere che animale è

presente in una foto, avremo un neurone di output che corrisponde all'animale "leone", uno per "criceto", uno per "pesce" e così via. Fra il livello di input e il livello di output ci sono uno o più livelli nascosti (hidden layer), che sono quelli dove avvengono le trasformazioni dei dati: ogni neurone di un livello è collegato a tutti i neuroni del livello immediatamente precedente e a tutti quelli del livello immediatamente successivo, i livelli sono quindi completamente interconnessi con quelli adiacenti. Questi collegamenti hanno un valore numerico chiamato peso, che definisce una misura di quanto importante sia il collegamento fra una specifica coppia di neuroni.

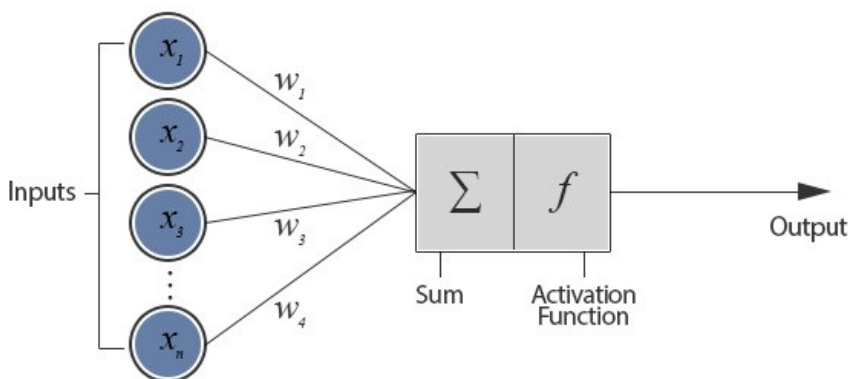


Figura 2.2: Rappresenzione di un neurone artificiale

Nell'immagine 2.2 è possibile osservare le componenti principali di un neurone artificiale, in particolare viene evidenziato come un neurone di in un livello intermedio riceva in ingresso dagli archi in un uscita di tutti i neuroni del livello precedente i vari input, ognuno col proprio peso. La prima operazione che viene effettuata è una sommatoria dei pesi moltiplicati per i vari input ricevuti al fine di ottenere un unico peso per quel determinato neurone, solitamente rappresentato con U .

$$U = \sum_{k=1}^n x_i \cdot w_i$$

Tale valore verrà in seguito passato alla funzione di attivazione e il risultato propagato al livello successivo.

Il funzionamento operativo appena descritto è chiamato *Feed Forward Propagation NN*, ovvero le connessioni tra le unità non formano cicli, differenziandosi dalle *Feedback Propagation NN*. Le informazioni si muovono solo in una direzione, partendo dai nodi d'ingresso, attraversando i nodi nascosti fino ad arrivare ai nodi di uscita. I neuroni ad ogni istante temporale non hanno cognizione degli input

ricevuti in passato ma hanno una visione dei soli dati attualmente in loro possesso. Da un punto di vista onnisciente, non accade altro che un continuo aggiornarsi di pesi per cercare di approssimare nel modo più accurato possibile la funzione che descrive genericamente il problema. Questo processo è chiamato *processo di apprendimento*.

Feedback Propagation Neural Network

Mentre si sta leggendo un testo, come ad esempio questo paragrafo, il cervello umano mantiene persistente nella memoria temporale le varie parole, ed elaborandole consente di comprendere il significato delle frasi durante la lettura.

Le reti neurali tradizionali non operano in questo modo, bensì ad ogni iterazione buttano via, ad eccezione dei pesi, qualsiasi informazione. Le reti neurali con *retro propagazione dell'errore* (*Feedback Propagation*) o *ricorrenti* adottano dei cicli affinché ad ogni iterazione l'output di un neurone non sia solamente l'input per il livello successivo per l'iterazione corrente ma anche l'input per se stesso all'iterazione successiva. Una RNN può essere pensata come più copie della stessa rete, ognuna delle quali invia messaggi alla successiva.

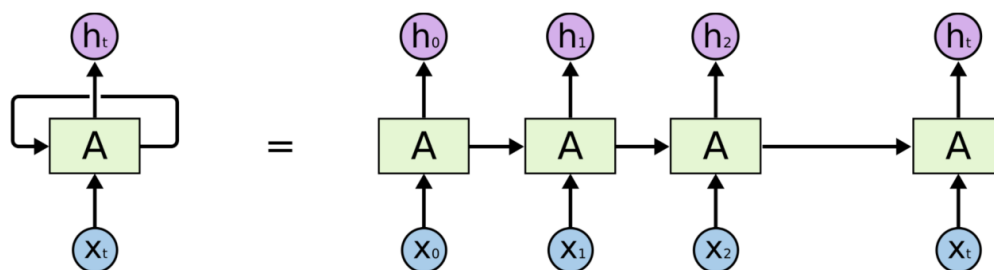


Figura 2.3: Rappresentazione di un neurone artificiale di una rete neurale ricorrente in forma compatta ed estesa

Osservando la figura 2.3 si nota subito la natura a catena intrinseca delle reti neurali ricorrenti, facilmente associabile a problemi che operano su sequenze e liste. Sono l'architettura naturale da utilizzare per problemi come il riconoscimento vocale, la modellazione linguistica, la traduzione e ovviamente per problemi di sequenze temporali come la previsione degli inquinanti.

2.2.3 Funzione di attivazione

Come anticipato, il risultato della sommatoria che precede la funzione di attivazione potrebbe essere qualsiasi valore, da $-\infty$ a $+\infty$, questo fa sorgere un problema di fondo per i neuroni dei livelli successivi: il neurone che fornisce l'input è attivo o no? La funzione di attivazione è una funzione il cui compito è regolare il valore in uscita prodotto da un neurone e definire se le connessioni esterne debbano considerare tale neurone attivo o meno. In letteratura ne sono stati definiti molti tipi, dalle più semplici alle più complesse; di seguito verranno illustrate le più note.

Funzione Heaviside o Threshold

La *funzione Heaviside*, più comunemente chiamata *funzione soglia*, è una delle funzioni più semplici in quanto possiede internamente una soglia superata la quale il risultato restituito sarà 1, altrimenti 0.

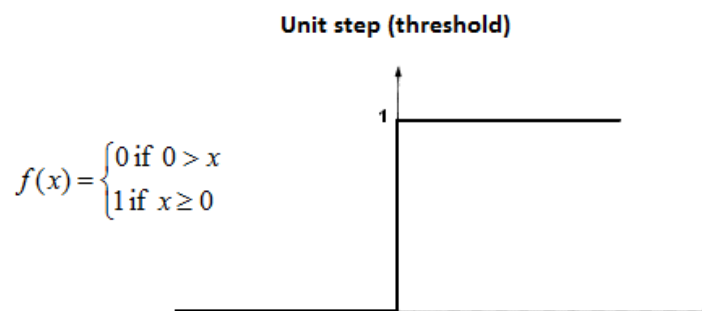


Figura 2.4: Funzione con threshold

Una funzione semplice può andar bene per problemi la cui generalizzazione è facilmente raggiungibile, tuttavia potrebbero insorgere alcuni problemi di convergenza; Immaginando uno scenario di classificazione può capitare che vi siano più neuroni di output attivi quando in realtà il risultato atteso è di un solo neurone di output attivo per singolo input. Dalla necessità di avere a disposizione una funzione che non sia binaria è stata definita la funzione lineare.

Funzione lineare

La funzione di attivazione lineare è una funzione in cui l'attivazione è proporzionale all'input fornito e non possiede un comportamento a scalino come la funzione precedente.

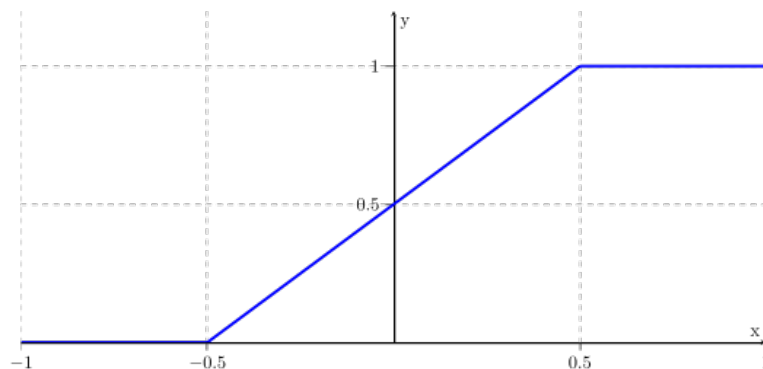


Figura 2.5: Funzione lineare

Purtroppo anche questa funzione presenta dei difetti, questi emergono soprattutto quando si ha a che fare con problemi molto complessi dove è necessario che la rete neurale sia composta da più livelli nascosti. Si assuma come scenario una rete neurale dove tutti i livelli sono configurati per utilizzare funzioni di attivazione lineari. Di livello in livello l'attivazione viene ricalcolata e passata al livello successivo come input, che verrà a sua volta ricalcolata mediante la stessa funzione nel livello successivo. Non importa quanti strati si definiscano, se tutti sono di natura lineare, la funzione di attivazione finale dell'ultimo livello non è altro che una semplice funzione lineare dell'input del primo livello. Ciò significa che questi N livelli possono essere sostituiti da un singolo livello. La flessibilità offerta dall'utilizzare più strati di neuroni viene annullata, indipendentemente da come questi vengano configurati. Una combinazione di funzioni lineari è a sua volta una funzione lineare.

Funzione Sigmoid

$$A(x) = \frac{1}{1 + e^{-x}}$$

Uno dei vantaggi di questa funzione è la sua natura non lineare, di conseguenza le combinazioni di questa funzione sono anch'esse non lineari rendendo possibile utilizzare più livelli nascosti.

Si noti in figura 2.6 come tra i valori X da -2 a 2, i valori Y corrispondenti sono molto ripidi, il che significa che qualsiasi piccola variazione nei valori di X in quella regione comporterà un cambiamento significativo nei valori di Y; la funzione ha la tendenza a portare i valori Y ad entrambe le estremità della curva, rendendola ottima per i problemi di classificazione.

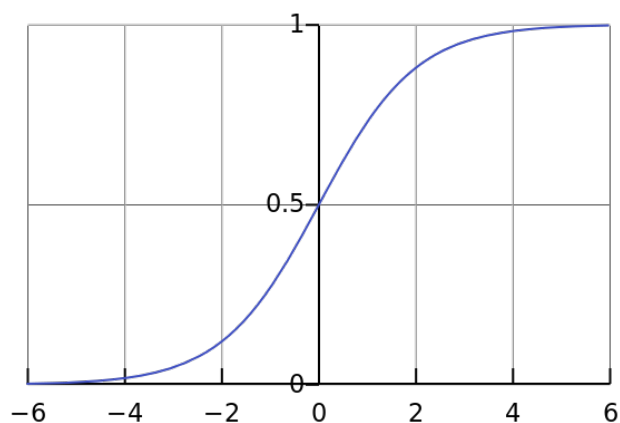


Figura 2.6: Funzione Sigmoid

Le funzioni sigmoidali sono le funzioni di attivazione più utilizzate ad oggi, tuttavia non sono esenti da problemi. Alle estremità la funzione tende ad essere molto meno reattiva a fronte di modifiche in X , questo comporta che il gradiente in quella regione sarà molto basso dando origine al *vanishing gradient problem*: la rete neurale rifiuta di apprendere ulteriormente o è drasticamente lenta.

Funzione Rectifier Linear Unit (ReLU)

$$A(x) = \max(0, x)$$

La funzione ReLU restituisce il valore passato in ingresso se positivo, altrimenti restituisce 0.

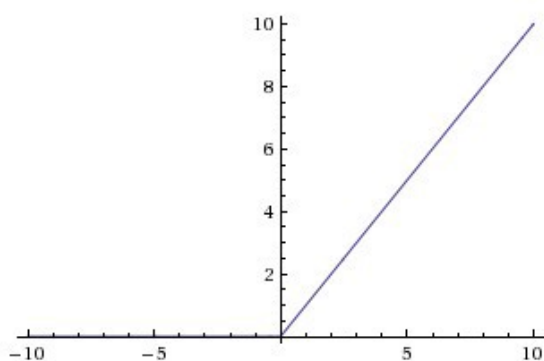


Figura 2.7: Funzione con ReLU

A prima vista questa funzione sembrerebbe avere gli stessi problemi della lineare ma sorprendentemente ReLU è di natura non lineare e le sue combinazioni sono anch'esse non lineari; questo comporta la possibilità di definire più livelli combinati tra loro. Una caratteristica tipica di ReLU è quella di limitare l'attivazione nel numero maggiore possibile di neuroni. In una rete neurale di grandi dimensioni con molti dati, usare Sigmoid farà sì che quasi tutti i neuroni si attivino e l'output della rete dipenda da gran parte di essi. Uno scenario del genere viene descritto come una rete con attivazione densa e di conseguenza onerosa; ReLU opera in modo tale che pochi neuroni nella rete si attivino, rendendo così le attivazioni sparse ed efficienti e la rete stessa più leggera.

ReLU è computazionalmente più efficiente rispetto a sigmoid non solo per il modo in cui opera ma anche perché si avvale di operatori matematici più semplici da elaborare; in un contesto di reti neurali a più livelli la differenza in termini di tempo è considerevole.

ReLU è perfetto? Per valori negativi di X il corrispondente valore Y viene convertito a 0, questo comporta che il gradiente diventi 0 anch'esso. Per le attivazioni in quella regione, i pesi non verranno aggiornati e i neuroni che entrano in questo stato smetteranno di rispondere alle variazioni di input; tale problema è chiamato *dying ReLU*. Vi sono varianti che arginano il problema semplicemente lasciando che il gradiente sia diverso da zero e di recuperare durante iterazioni successive [26].

2.2.4 Fasi di apprendimento e suddivisione dei dati

Nell'ambito delle reti neurali si possono individuare tre fasi operative, ognuna con scopi ben precisi. La prima fase è quella di *training*, ovvero la fase in cui si forniscono i dati di input e i dati di output attesi e la rete ne osserva le relazioni, aggiornando i pesi dei neuroni. Nella seconda fase - *evaluating* - si valutano le performance ottenute fornendo alla rete solamente i dati in input, chiedendole di fornire risultati; Tali risultati vengono confrontati coi dati di output già in possesso per valutare le performance. La terza fase di *prediction* è quella più interessante in quanto si opera su dati nuovi e si richiede alla rete di effettuare previsioni.

Ad ogni fase vengono forniti i dati corrispondenti, il dataset è suddiviso in due blocchi principali detti *training set* e *test set*. Il primo viene utilizzato per allenare la rete neurale, imparare dai dati le loro relazioni e valutarne la prestazioni dell'allenamento, mentre il secondo blocco viene sfruttato per predire dei risultati reali.

Il *training set* viene a sua volta suddiviso in due insiemi, il primo per l'allenamento vero e proprio mentre il secondo, chiamato *validation set* è utilizzato

per verificare che l'allenamento non presenti problemi di overfitting o underfitting (vedi 2.8). Tipicamente viene effettuata una suddivisione in 80% per il training e 20% per la validazione.

2.2.5 Overfitting e Underfitting

La sfida principale nell'ambito del machine learning è garantire il funzionamento ottimale del modello utilizzato sui dati nuovi, ovvero mai visti da esso durante la fase di allenamento. Questo processo è chiamato generalizzazione.

Un buon modello di machine learning si pone come obiettivo la minimizzazione dell'errore sui dati di allenamento, chiamato *training error* e la minimizzazione dell'errore di generalizzazione o *test error*, che equivale all'errore previsto su dati di input nuovi [2].

Come anticipato, una buona quantità di dati è un presupposto principale per ottenere buoni risultati, tuttavia, a seconda di come questi vengono suddivisi e forniti alla rete neurale possono far sorgere due problemi noti come *Overfitting* e *Underfitting*.

Overfitting

L'Overfitting occorre quando un modello impara troppo dai dati forniti durante l'allenamento risultando in due conseguenze negative. Per prima cosa un modello complesso richiederà un tempo sempre maggiore per essere eseguito. In secondo luogo, un modello complesso può ottenere ottimi risultati sui dati di training, e risultati inferiori su quelli di test, perché esso non sarà in grado di generalizzare, ma avrà imparato solamente le relazioni dei dati di training. In figura 2.8 abbiamo un esempio di overfitting: il grafico della funzione di loss decresce sia sul set di test che su quello di training, però oltre un certo punto (la soglia della capacità ottima), l'errore di generalizzazione inizia a crescere a causa dell'overfitting.

Per risolvere il problema dell'overfitting è necessario un modo per esprimere la complessità di un modello. Considerato che un modello è semplicemente un vettore di pesi, si può esprimere la sua complessità indicando il numero di pesi diversi da zero, cosicché avendo 2 modelli $M1$ e $M2$ con le stesse performance a livello di loss, è opportuno scegliere il modello che ha il numero minore di pesi diversi da zero [1].

Underfitting

L'Underfitting occorre quando il modello non riceve abbastanza dati in input e non riesce a minimizzare sufficientemente l'errore di allenamento e a generalizzare dati nuovi. In generale un modello che soffre di underfitting è un modello inadatto.

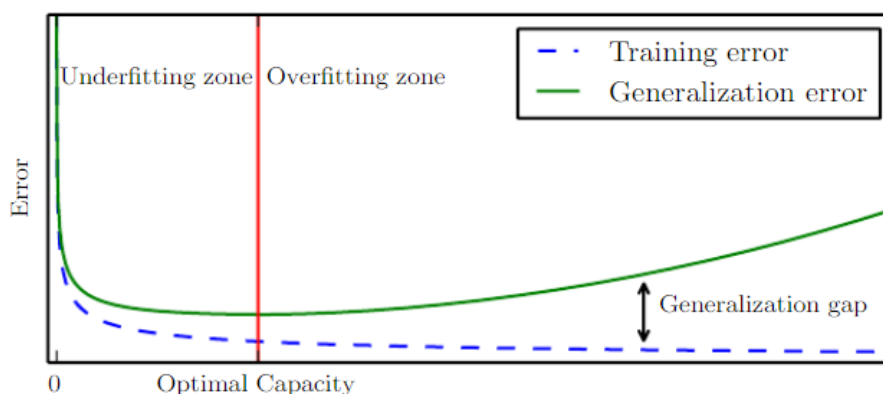


Figura 2.8: Relazione tra capacità ed errore

2.3 Configurazione della rete neurale

La rete neurale possiede svariate impostazioni che possono influenzare in maniera considerevole le sue performance, si possono suddividere in due macro categorie: parametri e iperparametri.

2.3.1 Parametri

Un parametro di un modello è una variabile di configurazione interna alla rete neurale, come ad esempio un peso interno ad un neurone; il suo valore può essere stimato in base ai dati. I parametri sono necessari per consentire al modello di effettuare predizioni e generalmente non sono impostati mai manualmente, bensì tramite un algoritmo di ottimizzazione che li determina dinamicamente durante le fasi di allenamento della rete neurale. Questo algoritmo si basa sull'ottimizzazione del gradiente.

2.3.2 Ottimizzazione del gradiente

Ottimizzare consiste nel minimizzare o massimizzare una funzione $f(x)$ alterando i valori di x . Solitamente tutti i problemi di ottimizzazione sono posti come minimizzazione di $f(x)$ e la massimizzazione è ottenuta minimizzando $-f(x)$. La funzione da minimizzare o massimizzare viene chiamata funzione obiettivo e quando si tratta di minimizzare è anche definita come *funzione di loss* o *funzione di errore*.

Supponiamo di avere una funzione $y = f(x)$ con x e y valori reali. La derivata di questa funzione si esprime con $f'(x)$ o $\frac{dy}{dx}$ e rappresenta il coefficiente angolare o la pendenza di $f(x)$ nel punto x . Può essere usata per specificare come ridimensionare

un piccolo cambiamento nell'input per ottenere un cambiamento corrispondente nell'output:

$$f(x + \epsilon) \approx f'(x) + \epsilon \cdot f(x)$$

Risulta comodo utilizzare la derivata per minimizzare una funzione perché ci spiega come cambiare x per ottenere un piccolo miglioramento su y . Per esempio, dato che $f(x - \epsilon \cdot \text{segno}(f'(x)))$ è minore di $f(x)$ per ϵ abbastanza piccoli, possiamo minimizzare $f(x)$ spostando x in piccoli passi del segno opposto della derivata. Questa tecnica è chiamata discesa del gradiente, mostrata in figura 2.9.

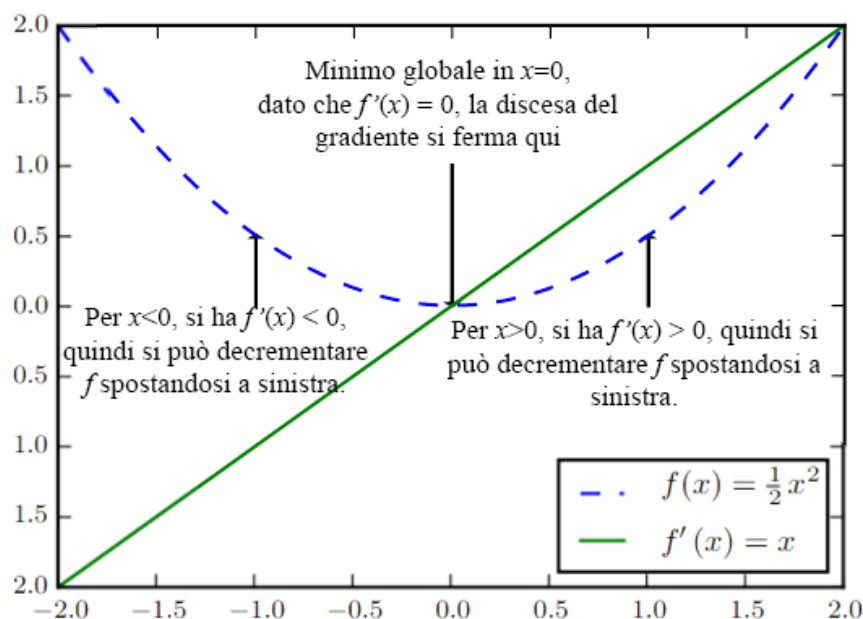


Figura 2.9: Funzionamento dell'algoritmo di discesa del gradiente

Quando $f'(x) = 0$ la pendenza è 0 e quindi la derivata non fornisce informazioni sulla direzione in cui muoversi. I punti in cui ciò succede sono detti punti critici, mostrati in figura 2.10, e possono essere di minimo locale, dove la funzione non può più decrescere per passi infinitesimali, di massimo locale, quando non può più crescere con passi infinitesimali oppure possono essere punti di sella, che non sono né massimi né minimi.

Il punto che ottiene il valore in assoluto più basso della funzione $f(x)$ è detto minimo globale. Può esserci un solo minimo globale o più di uno se equivalenti, mentre può esserci un minimo locale non ottimo globalmente. Nell'ambito del deep learning la sfida è ottimizzare una funzione con molti minimi locali non ottimali, punti di sella circondati da zone piatte e con input multidimensionali.

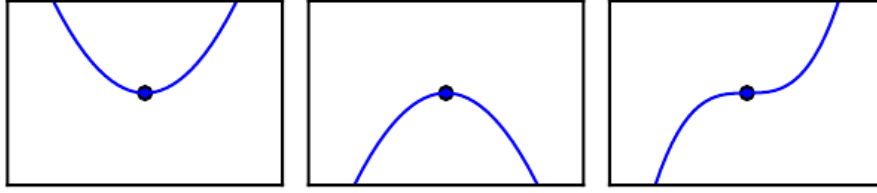


Figura 2.10: Minimo locale, massimo locale e punto di sella

Occorre quindi accontentarsi di trovare un valore di f basso, ma certamente non il minimo. Come già detto, spesso è necessario minimizzare funzioni con più input: $f : \mathbb{R}^N \rightarrow \mathbb{R}$ e perché il concetto di minimizzazione abbia senso deve esserci un solo output.

Per le funzioni con input multipli si utilizza il concetto di derivata parziale. La derivata parziale $\frac{\partial}{\partial x_i} f(x)$ misura il cambiamento di f quando solo la variabile x_i è incrementata nel punto x . Il gradiente è definito come il vettore che ha come componenti le derivate parziali di una funzione, $\nabla_x f(x)$. L'elemento i del gradiente è la derivata parziale di f rispetto ad x_i . In più dimensioni, i punti critici corrispondono ai punti in cui ogni elemento del gradiente vale 0. Sapendo che il gradiente negativo punta verso il basso, possiamo decrementare f muovendoci in direzione di esso.

Questo procedimento è detto metodo di discesa più ripida, o di discesa del gradiente. Esso propone un nuovo punto

$$x' = x - \epsilon \cdot \nabla_x f(x), \quad (2.1)$$

dove ϵ è il tasso di apprendimento, uno scalare positivo che determina la grandezza dello step.

L'algoritmo converge quando ogni elemento è 0 o prossimo ad esso ed in alcuni casi è possibile saltare direttamente al punto critico risolvendo l'equazione $\nabla_x f(x) = 0$ per x . Anche se la discesa del gradiente è limitata a spazi continui, il concetto di piccoli spostamenti ripetuti per ottenere un miglioramento può essere generalizzato su spazi discreti [2].

2.3.3 Varianti di Gradient descent

Esistono tre varianti principali dell'algoritmo di discesa del gradiente, che differiscono per la quantità di dati utilizzati per calcolare il gradiente della funzione obiettivo. In base alla quantità di dati cambia anche il tempo necessario per aggiornare i parametri e l'accuratezza di questo aggiornamento.

Batch gradient descent

È la tecnica di base, descritta in precedenza, che calcola il gradiente su tutto il dataset per effettuare un solo aggiornamento. Posti $J(\theta)$ la funzione obiettivo e θ i parametri del modello, la funzione 2.1 si può riscrivere come:

$$\theta = \theta - \epsilon \cdot \nabla_{\theta} J(\theta).$$

Questo approccio non permette di effettuare l'aggiornamento dei parametri del modello in tempo reale e può essere sconsigliato per dataset troppo grandi che non possono risiedere integralmente in memoria.

Stochastic gradient descent

L'algoritmo di discesa stocastica del gradiente (SGD) effettua un aggiornamento dei parametri per ogni variabile di input $x^{(i)}$ e variabile di output $y^{(i)}$:

$$\theta = \theta - \epsilon \cdot \nabla_{\theta} J(\theta; x^{(i)}; y^{(i)}).$$

Rispetto alla versione batch, che effettua calcoli ridondanti su tutto il dataset prima di ogni aggiornamento, SGD aggiorna i parametri ad ogni iterazione di allenamento.

Questo causa una fluttuazione della funzione obiettivo (mostrato in figura 2.11), che da un lato può portare ad un salto verso minimi locali migliori, dall'altro complica la convergenza verso il minimo globale, in quanto l'algoritmo tende ad oltrepassarlo.

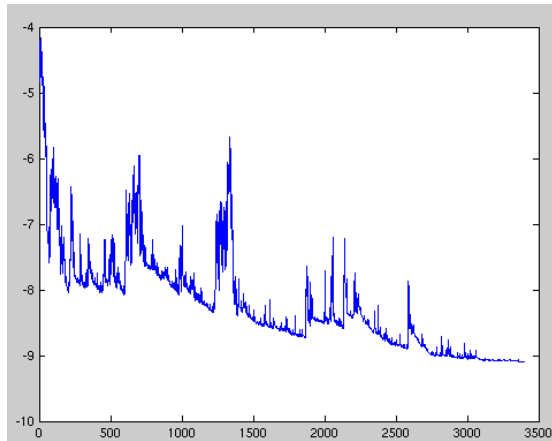


Figura 2.11: Fluttuazione della funzione obiettivo in SGD[28]

Il problema può essere evitato diminuendo gradualmente il tasso di apprendimento portando SGD alle stesse performance di convergenza della versione batch.

Mini-batch gradient descent

La versione mini-batch di gradient descent è una combinazione dei due algoritmi precedenti, viene eseguito un aggiornamento per ogni mini-batch di n elementi:

$$\theta = \theta - \epsilon \cdot \nabla_{\theta} J(\theta; x^{(i:i+n)}, y^{(i:i+n)}).$$

In primo luogo vengono ridotte le fluttuazioni della funzione obiettivo, portando ad una convergenza più regolare e utilizzate operazioni di calcolo più efficienti che evitano di tenere tutto il dataset in memoria [5].

2.3.4 Iperparametri

A differenza dei parametri, un iperparametro è una variabile esterna al modello, il cui valore non viene determinato in base ai dati forniti in input durante il training. Vanno specificati prima di iniziare l'apprendimento e, dato che non sono riassumibili da una funzione, vengono spesso impostati manualmente dall'utilizzatore in modo sperimentale e iterativo. Fortunatamente esistono tecniche di ottimizzazione automatica, trattate nel capitolo 6.

La comprensione dei vari iperparametri e delle conseguenze che porta una loro modifica sono fondamentali per configurare all'ottimo il modello della propria rete neurale [13].

Number of epochs

Il numero di epoche è uno dei parametri più importanti, esso definisce il numero di volte in cui deve essere fornito alla rete neurale l'intero training set. A prima vista potrebbe non avere senso passare più e più volte gli stessi dati alla rete neurale, tuttavia è doveroso ricordare che si sta utilizzando una parte del data set completo e per ottimizzare l'apprendimento si sta utilizzando l'algoritmo *Gradient Descent* il cui processo è iterativo. Pensare di riuscire ad aggiornare correttamente i pesi di tutti i neuroni in una sola iterazione non è realizzabile. Mano a mano che le epoche passano i pesi vengono aggiornati, aggiustando la curva che approssima il problema da una condizione di underfitting ad una ottimale (evitando di arrivare all'overfitting). Volendo generalizzare, più i dati sono complessi e diversi tra loro, maggiore dovranno essere le epoche.

Batch Size

Quando il dataset è di dimensioni notevoli, anche con elaboratori molto performanti non è possibile all'interno di un'epoca osservare tutti i dati direttamente. E' possibile suddividere i dati di training in sottoinsiemi di dimensione prefissata

chiamati *batch*, anche parzialmente sovrapposti. Questo diminuisce notevolmente l'utilizzo della memoria centrale ma peggiora le performance, in termini di tempo, nella fase di apprendimento.

Step per epoch

Nel caso in cui si sia suddiviso il set in batch è importante che venga impostato anche il numero di passi per epoca. Considerato che tutti i dati devono essere visionati almeno una volta è necessario che questo parametro sia calcolato correttamente. Si assuma di avere 20000 immagini e impostato la batch size a 100, gli step per epoca devono necessariamente essere $20000/100 = 200$. Sorge spontanea una domanda: se può essere calcolato così facilmente, perchè non viene ricavato dalla rete in autonomia? Nonostante la letteratura suggerisca di seguire questa formula, viene mantenuta aperta la possibilità di numerosi esperimenti; uno di questi consiste nel mescolare casualmente l'ordine dei batch e prendere in considerazione qualche esempio in meno o passando più volte sullo stesso, variando leggermente questo parametro.

Learning rate (LR)

Il tasso di apprendimento è uno dei parametri più delicati da trattare in quanto una piccola variazione può danneggiare in modo significativo il processo di training. Il *LR* definisce quanto velocemente una rete debba abbandonare le conoscenze acquisite nel passato e sostituirle alle nuove.

Se ad un bambino vengono mostrati 10 gatti e tutti possiedono il pelo arancione penserà che i gatti abbiano tutti il pelo di quel colore e si aspetterà questa caratteristica anche quando cercherà di identificare un gatto mai visto in precedenza. Nel momento in cui vedrà un gatto nero e i suoi genitori gli diranno che è anch'esso un gatto (apprendimento supervisionato), con un tasso di apprendimento alto si renderà conto subito che il pelo arancione non è la caratteristica più importante dei gatti, altrimenti, con un tasso di apprendimento molto basso, penserà che quel gatto nero è anomalo (outlier) e che i gatti sono ancora tutti arancioni.

Più è alto il tasso di apprendimento più la rete cambia idea rapidamente. Questo può essere positivo nel caso del bambino, ma può anche avere risvolti negativi. Se il tasso di apprendimento è troppo alto, il bambino potrebbe iniziare a pensare che tutti i gatti siano neri anche se ha visto più gatti arancioni che neri.

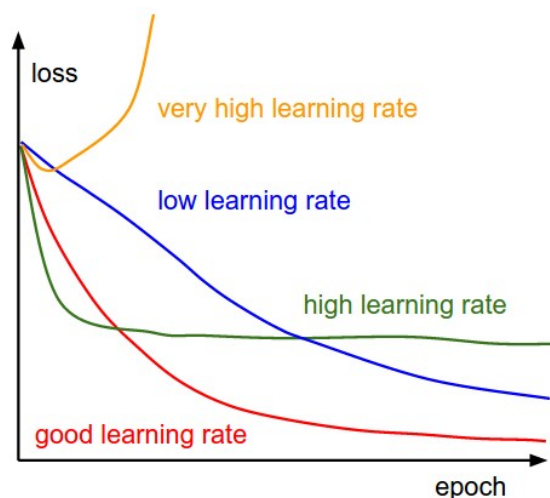


Figura 2.12: Possibili andamenti a seconda del LR

In generale, si desidera trovare un tasso di apprendimento sufficientemente basso da consentire alla rete di convergere in qualcosa di utile, ma abbastanza alto da non dover impiegare giorni per addestrarsi [29].

Number of Hidden Layers

Questo parametro definisce il numero di livelli nascosti, ovvero quanti livelli includere tra lo strato di input e di output. Per la maggior parte dei problemi un solo livello nascosto è più che sufficiente, per problemi più complessi come quello che si sta trattando uno solo non basta per generalizzare con risultati soddisfacenti. E' doveroso specificare come la complessità computazionale è linearmente dipendente con il numero dei livelli nascosti; a fronte di una rete con due livelli nascosti è naturale aspettarsi che il tempo necessario per il training sia circa il doppio di quello richiesto per una rete con un solo livello nascosto. Questo è da considerarsi vero solamente nel caso in cui i due livelli siano identici nella configurazione, tuttavia è usuale dimezzare di livello in livello il numero di unità neurali.

Number of Hidden Unit

Il numero di unità nascoste definisce quanti neuroni debbano essere presenti in uno specifico livello. Come anticipato è bene assicurarsi che i livelli nascosti abbiano un numero di neuroni via via inferiore per ogni livello, in direzione del livello di output.

Non esiste un modo specifico per trovare il valore ottimale di questo parametro in ogni strato, nonostante ciò la letteratura moderna consiglia di utilizzare numeri compresi tra la dimensione dell'input e quella dell'output.

Prendendo in esame il problema dell'inquinamento, a fronte delle 42 stazioni di misurazione, ognuna con la rilevazione di 6 tipi di misurazione, si ottiene un input con una dimensione di $(42 \times 6, T) = (252, T)$ dove T è il numero totale di rilevazioni orarie. L'output sarà il valore di NO_2 delle 42 stazioni. Di conseguenza, il numero di neuroni dovranno essere rispettivamente di 252 per il primo livello, di 42 per l'ultimo e per tutti i livelli nascosti un numero compreso in quell'intervallo. Solitamente si dimezza il valore per ogni livello attraversato.

Dropout

Il Dropout consiste nell'ignorare durante la fase di addestramento, alcune unità di un certo insieme di neuroni che viene scelto casualmente. Con ignorare si intende come queste unità non siano più considerate durante un particolare passaggio in avanti o indietro dei valori che si scambiano i neuroni. Nello specifico, ad ogni iterazione di addestramento i singoli neuroni vengono o abbandonati dalla rete con probabilità $1 - p$ o mantenuti con probabilità p , in modo tale da mantenere attiva una rete ridotta; Naturalmente anche i collegamenti in entrata e in uscita verso un nodo abbandonato vengono rimossi per notificare il suo stato agli altri.

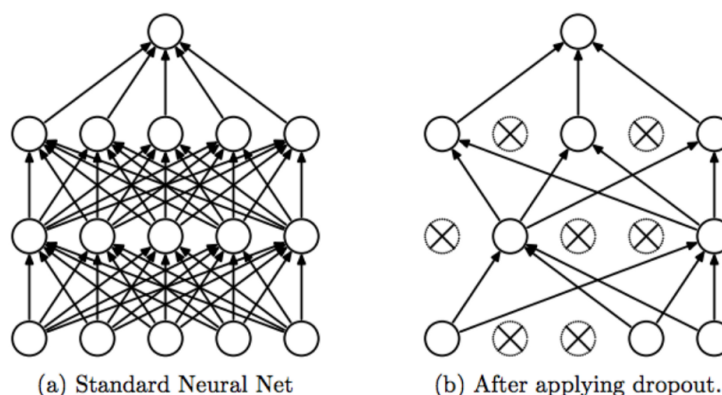


Figura 2.13: Confronto di una rete neurale prima e dopo l'applicazione del dropout

Lo scopo per il quale è stato definito il meccanismo alla base del dropout è quello di prevenire l'overfitting; uno strato completamente connesso contiene la maggior parte dei parametri e i neuroni al suo interno sviluppano delle co-dipendenze tra loro durante l'allenamento, frenando la potenza individuale di ogni singolo neurone e portando ad un eccessivo adattamento ai dati di allenamento [15].

2.4 Funzioni di errore

La funzione di loss o funzione di errore esprime l'errore di generalizzazione che la rete commette sulle predizioni. Come descritto in precedenza lo scopo dei vari algoritmi di ottimizzazione del gradiente è minimizzare questa funzione, per migliorare le performance del modello. Ne esistono varie tipologie, più o meno adatte in base al tipo di problema, di seguito verranno trattate quelle relative ai problemi di regressione.

2.4.1 Mean square error

La funzione Mean square error (MSE), detta anche L2 loss è la media quadratica dell'errore su tutto il dataset. Ponendo $h_\theta(x_i)$ il valore predetto e y_i il valore reale, si esprime come:

$$J = \frac{1}{n} \sum_{i=1}^n (y_i - h_\theta(x_i))^2.$$

Per la sua natura quadratica l'errore L2 è suscettibile a valori anomali e causa un eccessivo adattamento del modello a questi valori. Tuttavia risulta più semplice da calcolare rispetto alla L1 loss, trattata di seguito.

2.4.2 Mean absolute error

Il Mean absolute error (MAE) o L1 loss, minimizza le differenze assolute tra i valori stimati e quelli reali.

$$J = \sum_{i=0}^n |y_i - h(x_i)|$$

Generalmente è meno influenzata dai valori anomali rispetto a MSE, tuttavia presenta il problema di un gradiente costante, che può raggiungere valori alti anche per valori piccoli di errore. Per risolvere questo è necessario usare un tasso di apprendimento dinamico [27].

2.4.3 Huber loss

Huber Loss è una funzione quadratica per piccoli valori di errore e lineare per valori più grandi. Il valore dell'errore per renderla quadratica è regolato da un iperparametro δ :

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{per } |y - f(x)| \leq \delta \\ \delta |y - f(x)| - \frac{1}{2}\delta^2 & \text{altrimenti.} \end{cases}$$

La scelta di δ determina quale valore considerare anomalo e quale no, in modo tale che valori più grandi siano minimizzati con L1 e quelli più piccoli con L2, maggiormente sensibile. Usando Huber loss si risolvono i problemi più rilevanti presenti nelle funzioni precedentemente trattate, al costo di dover configurare un iperparametro δ ulteriore [19].

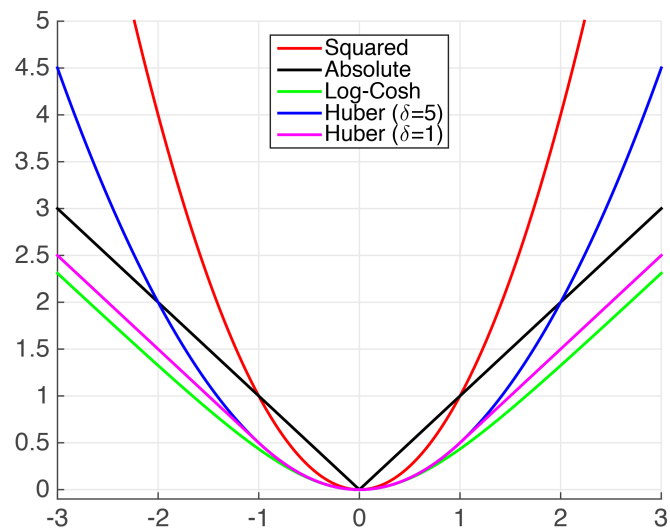


Figura 2.14: Confronto tra le varie funzioni di errore [25]

Capitolo 3

Progettazione e Sviluppo

Dopo aver ampiamente trattato in termini teorici l'intelligenza artificiale nel capitolo precedente, in questo capitolo verranno descritti gli strumenti utilizzati e i modelli configurati per la definizione di una rete neurale in grado di predire i valori dell'inquinamento di NO_2 in città.

3.1 Tecnologie disponibili

All'interno di questa sezione sono raccolti gli strumenti individuati per la realizzazione di questo progetto. Nonostante tali strumenti siano disponibili per vari linguaggi di programmazione, è stato scelto il Python in quanto linguaggio nativo di alcune librerie e per la presenza di moduli già pronti per l'import dei dati realizzati da ex colleghi.

3.1.1 Tensorflow

Tensorflow è una libreria software open source per il calcolo numerico che si avvale di grafi come architettura per il flusso di dati. I nodi del grafo eseguono operazioni matematiche, mentre gli archi trasportano le matrici di dati multidimensionali (tensori). Avvalendosi di un'architettura molto flessibile, Tensorflow consente di distribuire il calcolo su una o più CPU o GPU in un desktop, server o dispositivo mobile senza dover modificare il codice, aumentandone la scalabilità. TensorFlow include anche TensorBoard, un kit di strumenti per la visualizzazione dei dati.

TensorFlow è stato originariamente sviluppato da ricercatori e ingegneri del team Google Brain per il progetto Google's Machine Intelligence Research allo scopo di condurre la ricerca sull'apprendimento automatico e sulle deep neural

net. Il sistema è sufficientemente generale da poter essere applicabile anche in una vasta gamma di altri domini [8].

3.1.2 Keras

Nonostante la flessibilità offerta da Tensorflow il suo utilizzo risulta assai complesso e le sue API poco intuitive; Keras è una libreria per lo sviluppo rapido di reti neurali di alto livello, scritta in Python e che basa il suo funzionamento sul backend di TensorFlow. È stato sviluppato con l'idea di consentire una rapida sperimentazione, ovvero consentire di passare dall'idea al risultato nel minor tempo possibile. Keras consente una prototipazione veloce grazie alla facilità d'uso, alla modularità ed estensibilità che garantisce. Supporta reti convoluzionali, ricorrenti e combinazioni di entrambe, oltre a poter essere eseguita sia su CPU che GPU [18].

3.1.3 Numpy e Pandas

Numpy è uno dei package fondamentali per il calcolo scientifico in Python ed è essenziale per utilizzare Keras. Esso fornisce un array multi-dimensionale e svariate funzioni per manipolare i dati e gli array stessi [4].

Pandas è una libreria di alto livello per interagire con dati relazionali o etichettati. Fornisce due strutture dati di base: serie (una dimensione) e dataframe (due o più dimensioni) insieme alle funzioni ottimizzate per manipolarle. È stata sfruttata principalmente per la fase di importazione e riorganizzazione dei dati, oltre che per l'imputazione dei valori mancanti [3].

3.2 Ricerca della funzione di interpolazione ottimale

Nel capitolo 1 è stato accennato come per il riempimento dei dati mancanti sia stato utilizzata l'interpolazione lineare; Keras include svariate funzioni di interpolazione che vanno confrontate utilizzando i dati a propria disposizione per verificare quale sia quella ottimale. Nella pratica le funzioni sono state confrontate per ogni stazione ma per semplicità di trattazione in figura 3.1 viene mostrato il risultato di una sola stazione in quanto, fortunatamente, tutte le altre si sono comportate in modo analogo. Nel caso contrario sarebbe stato necessario effettuare una scelta tra quelle migliori.

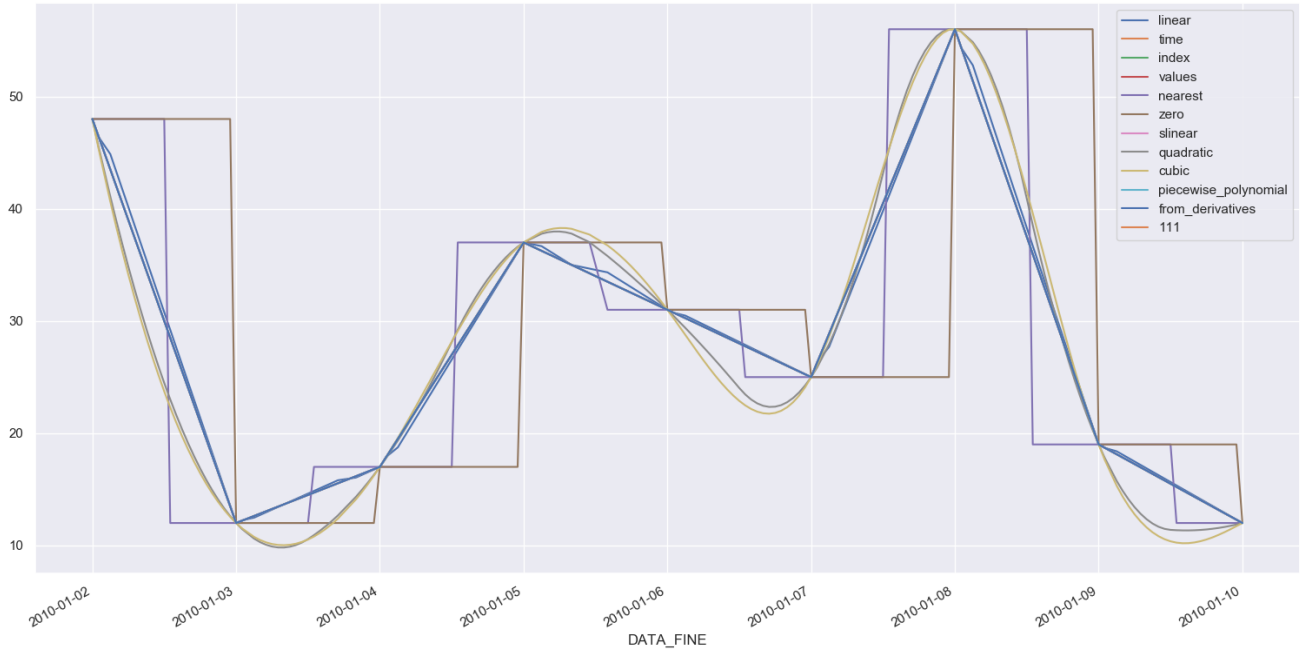


Figura 3.1: Confronto funzioni di interpolazione per la stazione 200003 sui valori di NO_2

Dal grafico è possibile notare in modo evidente come la funzione *cubica* e *quadratica* abbiano un comportamento a 'scalino' con un'approssimazione grossolana e inadatta al contesto. Le funzioni *polinomiale a tratti* e *derivata* effettuano un ottimo lavoro di approssimazione smorzando gli errori e le imperfezioni, tuttavia le due curve risultano molto distanti dai valori iniziali e per paura che questo potesse influenzare i risultati sono state scartate. Tutte le altre funzioni si comportano in modo pressoché equivalente e la scelta naturale è stata quella di utilizzare la più efficiente: *l'interpolazione lineare*. E' necessario specificare come la funzione *time* sia una funzione identica a quella *lineare* se non per il fatto che viene tenuto in considerazione anche l'asse orizzontale del tempo; considerando che il tempo è nel problema delle previsioni un elemento chiave, a livello implementativo è stata utilizzata tale funzione, da considerarsi identica a quella lineare.

3.3 Funzioni per la valutazione e il confronto dei modelli

3.3.1 Funzione di errore

Le principali funzioni a disposizione per la valutazione dei modelli sono quelle elencate nel paragrafo 2.4. Dato che la scelta della funzione di errore deve essere presa tenendo in considerazione dei dati a disposizione, si è scelto di utilizzare la *mae* perché essa è quella consigliata in letteratura per grandi dataset che presentano molto rumore.

3.3.2 Ottimizzatore

Come ottimizzatore è stato scelto Adam, un'alternativa al classico SGD (vedi paragrafo 2.3.3) che mantiene un learning rate differente per ogni singolo parametro, combinando i vantaggi di due algoritmi di ottimizzazione (RMSProp e AdaGrad) ed operando in maniera computazionalmente efficiente [12].

3.4 Struttura dei dati

In seguito alle trasformazioni effettuate nel capitolo 1 e grazie alla libreria di pandas è possibile rappresentare i dati in modo strutturato attraverso i *DataFrame*. I *DataFrame* sono degli array multidimensionali che offrono attraverso interfacce semplificate l'accesso ai dati in modo agevole. In figura 3.2 vengono rappresentati i dati così come sono stati utilizzati per singola stazione.

Station	7000014												Time		
Parameter	2	1000	1001	1003	1004	100500	100501	...	100505	100506	100507	100508	Month	DayOfWeek	Hour
DateTime								...							
2012-12-31 23:00:00	46.0	276.55	0.0	77.0	3.5	0	0	...	1	0	0	0	12	0	23
2013-01-01 00:00:00	43.0	276.45	0.0	77.0	4.1	0	0	...	1	0	0	0	1	1	0
2013-01-01 01:00:00	50.0	276.15	0.0	78.0	4.0	0	0	...	1	0	0	0	1	1	1
2013-01-01 02:00:00	32.0	276.45	0.0	78.0	3.4	0	0	...	1	0	0	0	1	1	2
2013-01-01 03:00:00	35.0	275.95	0.0	76.0	1.1	0	0	...	1	0	0	0	1	1	3

Figura 3.2: Struttura dei dati (singola stazione) con *ohe* sulla direzione del vento

Tabella 3.1: Legenda dei parametri

<i>id</i>	<i>parametro</i>
2	biossido d'azoto
1000	temperatura
1001	precipitazioni
1003	umidità
1004	velocità del vento
1005	direzione del vento

3.5 Definizione dei modelli

In questa sezione verranno elencati i modelli utilizzati per la sperimentazione, descrivendo per ognuno l'architettura e mostrandone l'implementazione.

3.5.1 Modelli lineari

I modelli lineari sono i più semplici e definiscono una rete neurale densamente connessa. Il primo livello deve essere necessariamente *Flattern* il cui compito è quello di appiattire l'input; questo perchè i livelli lineari richiedono che l'input sia unidimensionale. Il livello lineare *Dense* opera nel modo tradizionale dei neuroni seguendo la formula $output = attivazione(punto(input, kernel) + bias)$ già vista in precedenza.

Semplice

```
model = Sequential()
model.add(layers.Flatten(input_shape=(sequence_length, x_data.shape[1])))
model.add(layers.Dense(num_y_signals, activation='linear'))
```

Figura 3.3: Implementazione modello lineare semplice

Il modello lineare semplice non presenta alcun livello nascosto pertanto il layer di input è direttamente connesso a quello di output. La funzione di attivazione utilizzata è quella lineare che rende densamente connessa la rete realizzata adattando la dimensione dell'input alla dimensione dell'output. E' sicuramente un modello che richiede poco tempo per l'addestramento.

Hidden layer singolo

```
model = Sequential()  
model.add(layers.Flatten(input_shape=(sequence_length, x_data.shape[1])))  
model.add(layers.Dense(hidden_layer_size, activation='relu'))  
model.add(layers.Dense(num_y_signals, activation='sigmoid'))
```

Figura 3.4: Implementazione modello lineare con singolo hidden layer

Al modello precedente viene aggiunto un layer intermedio con funzione di attivazione *ReLU* e modificata la funzione di attivazione di output da *lineare* a *sigmoid*. Nonostante le funzioni di attivazione non siano lineari il modello rimane densamente connesso in quanto vengono utilizzati layer *Dense*.

Hidden layer doppio

```
model = Sequential()  
model.add(layers.Flatten(input_shape=(sequence_length, x_data.shape[1])))  
model.add(layers.Dense(hidden_layer_size, activation='relu'))  
model.add(layers.Dense(hidden_layer_size, activation='relu'))  
model.add(layers.Dense(num_y_signals, activation='sigmoid'))
```

Figura 3.5: Implementazione modello lineare con duplice hidden layer

In questo modello il layer nascosto viene duplicato. E' da considerarsi un modello *dummy* in quanto, come accennato nel capitolo 2, due layer configurati in modo identico, ed in particolare con lo stesso numero di unità neurali e stessa funzione di attivazione, operano come se fossero un layer unico producendone gli stessi risultati ma in modo meno efficiente.

3.5.2 Modelli Gated Recurrent Unit (GRU)

I modelli basati su celle GRU derivano dai modelli LSTM e si differenziano principalmente perchè sono più semplici e performanti. Nelle celle GRU viene introdotto un gate di aggiornamento (concettualmente simile ad un interruttore) che decide se far passare o meno le informazioni alla cella successiva. Aggiungere dei gate ad un neurone equivale a sfruttare ulteriori operazioni matematiche con ulteriori set di pesi.

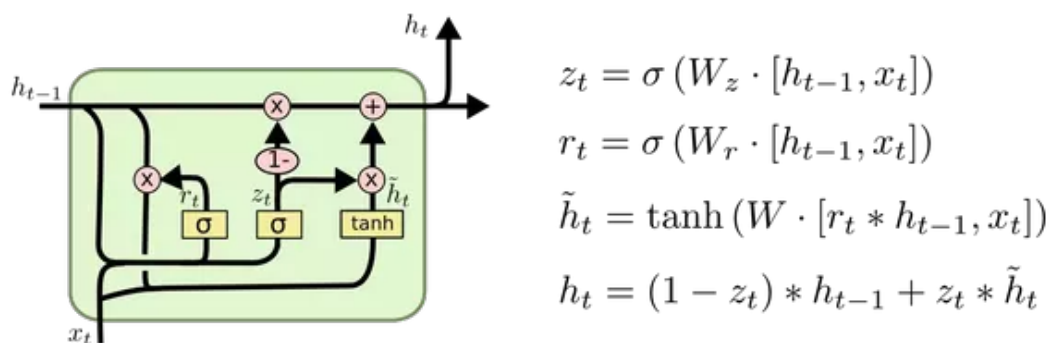


Figura 3.6: Architettura neurone GRU

Si è scelto di provare ad usare queste celle in quanto i relativi modelli sono più semplici da utilizzare e si allenano più velocemente rispetto ad LSTM.

Layer ricorrente con GRU

```
model = Sequential()
model.add(layers.GRU(hidden_layer_size, activation='relu', input_shape=(sequence_length, x_data.shape[1])))
model.add(layers.Dense(num_y_signals, activation='sigmoid'))
```

Figura 3.7: Implementazione modello ricorrente GRU semplice

Questo modello non possiede layer nascosti ed è la configurazione base per utilizzare le Gate Recurrent Unit. Il livello di output deve essere *Dense* in quanto alcuni neuroni di GRU, per il modo in cui opera, potrebbero essere 'disattivati' ed in output per motivi di libreria questo non è consentito.

Layer ricorrente con GRU e Dropout

```
model = Sequential()
model.add(layers.GRU(hidden_layer_size, activation='relu', input_shape=(sequence_length, x_data.shape[1])))
model.add(Dropout(0.2))
model.add(layers.Dense(num_y_signals, activation='sigmoid'))
```

Figura 3.8: Implementazione modello ricorrente GRU con dropout

Al modello precedente viene attivato il dropout con 20% di scarto; a livello implementativo opera come un hidden layer nonostante a livello logico sia solamente una funzione che opera sugli archi in uscita dai neuroni del primo livello verso quelli del livello di output.

Layer ricorrente con GRU e singolo hidden layer

```
model = Sequential()
model.add(layers.GRU(64, activation='relu', return_sequences=True, input_shape=(sequence_length, x_data.shape[1])))
model.add(layers.GRU(32, activation='relu'))
model.add(layers.Dense(num_y_signals, activation='sigmoid'))
```

Figura 3.9: Implementazione modello ricorrente GRU con singolo hidden layer

Al modello base di GRU viene aggiunto un livello nascosto della stessa tipologia. Al contrario del modello di prova definito nel macro-gruppo dei lineari, qui il numero di unità neurali viene dimezzato nel livello nascosto consentendo quindi una progressiva semplificazione dei dati.

Layer ricorrente con GRU e doppio hidden layer

```
model.add(layers.GRU(64, activation='relu', return_sequences=True, input_shape=(sequence_length, x_data.shape[1])))
model.add(layers.GRU(32, activation='relu', return_sequences=True))
model.add(layers.GRU(16, activation='relu'))
model.add(layers.Dense(num_y_signals, activation='sigmoid'))
```

Figura 3.10: Implementazione modello ricorrente GRU con duplice hidden layer

Al fine di verificare se l'aumento di hidden layer potesse portare a benefici è stato definito questo modello con un ulteriore livello nascosto ed un numero di unità neurali dimezzato ulteriormente (16).

3.5.3 Modelli Long Short-Term Memory (LSTM)

LSTM è il padre di tutte le architetture ricorrenti ed è anche per questo che è il meno performante. Nelle celle LSTM sono presenti ben due porte in più rispetto a GRU (Forget e Output) che rallentano il processo di apprendimento.

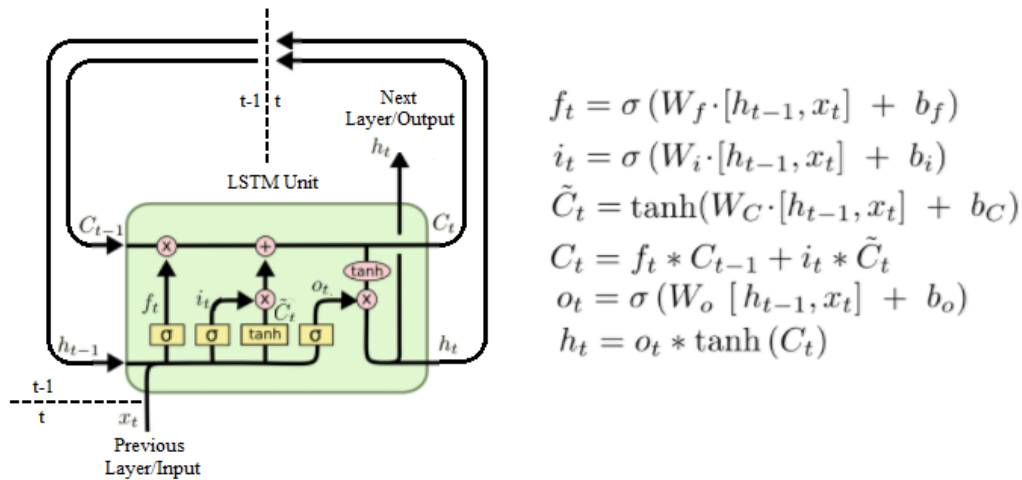


Figura 3.11: Architettura neurone LSTM

Sorge naturale chiedersi: perchè prenderlo in considerazione? LSTM avendo più gate dovrebbe riuscire a ricordare sequenze più lunghe delle celle GRU e superarle in compiti che richiedono la modellazione delle relazioni che si evidenziano nel lungo periodo, questo verrà verificato nelle successive sperimentazioni.

Layer ricorrente con LSTM

```
model.add(layers.LSTM(hidden_layer_size, activation='relu', input_shape=(sequence_length, x_data.shape[1])))
model.add(layers.Dense(num_y_signals, activation='sigmoid'))
```

Figura 3.12: Implementazione modello ricorrente LSTM semplice

In figura 3.12 è possibile osservare la configurazione base per l'utilizzo delle reti con unità LSTM. Similarmente a GRU è necessario il livello *Dense* perchè a causa dei *forget gate* non tutti gli archi in uscita dal primo livello potrebbero essere attivi e per vincoli di libreria questo non è consentito.

Layer ricorrente con LSTM e Dropout

```
model.add(layers.LSTM(hidden_layer_size, activation='relu', input_shape=(sequence_length, x_data.shape[1])))
model.add(Dropout(0.2))
model.add(layers.Dense(num_y_signals, activation='sigmoid'))
```

Figura 3.13: Implementazione modello ricorrente LSTM semplice con dropout

Al modello precedente viene attivato il dropout con scarto del 20%; si preme ricordare che il dropout non è un livello ma una funzione operante sugli archi in uscita dal livello da cui è preceduta.

Layer ricorrente con LSTM e singolo hidden layer

```
model.add(layers.LSTM(64, activation='relu', return_sequences=True, input_shape=(sequence_length, x_data.shape[1])))
model.add(layers.LSTM(32, activation='relu'))
model.add(layers.Dense(num_y_signals, activation='sigmoid'))
```

Figura 3.14: Implementazione modello ricorrente LSTM con singolo hidden layer

Un modello LSTM base a cui viene aggiunto un layer ricorrente nascosto della stessa tipologia con unità neurali dimezzate rispetto a quelle del livello di input, rispettivamente 32 e 64.

Layer ricorrente con LSTM e doppio hidden layer

```
model.add(layers.LSTM(64, activation='relu', return_sequences=True, input_shape=(sequence_length, x_data.shape[1])))
model.add(layers.LSTM(32, activation='relu', return_sequences=True))
model.add(layers.LSTM(16, activation='relu'))
model.add(layers.Dense(num_y_signals, activation='sigmoid'))
```

Figura 3.15: Implementazione modello ricorrente LSTM con duplice hidden layer

Analogamente al caso GRU, si è provato ad aggiungere un ulteriore layer nascosto LSTM con unità neurali dimezzate (16) per verificare l'eventuale presenza di benefici. Tra quelli proposti in questo capitolo è il modello più complesso e computazionalmente più oneroso.

Capitolo 4

Sperimentazione e Analisi dei risultati

In questo capitolo verranno mostrati i risultati raggiunti con i modelli definiti nel capitolo precedente e messi a confronto attraverso la funzione di valutazione *mae* scelta in precedenza.

4.1 Valutazione dei modelli

In questa sezione verranno elencati per i vari modelli utilizzati la differenza di loss in fase di training e validazione ed il risultato della predizione.

Di seguito viene indicata la configurazione di default utilizzata per tutti i modelli, ricavata sperimentalmente in modo trial and error senza avvalersi di ottimizzatori di iperparametri.

- batch size: 128
- epoche: 100
- train-test split: 90%–10%
- train-validation split: 90%–10%
- hidden layer: 32 unità di default (ove non specificato)

Nonostante siano state impostate 100 epoche in tutti i test queste non sono mai state sfruttate per intero, questo grazie alle callback disponibili in Keras. Le più rilevanti sono *EarlyStopping*, che interrompe il training se per 5 volte consecutive la loss non viene ulteriormente minimizzata e la callback *ReduceLROnPlateau*

che interviene in maniera meno drastica della precedente, riducendo il tasso di apprendimento dopo 3 epoche in cui la loss non diminuisce. A seguito delle callback appena definite si rende necessaria anche la definizione del *Model Checkpoint* che si occupa di salvare al termine di ogni epoca i pesi ottimali ed evita che si utilizzino quelli di un epoca che ha causato lo stop preventivo del training.

```
callback_early_stopping = EarlyStopping(monitor='val_loss',
                                         patience=4, verbose=1)

callback_reduce_lr = ReduceLROnPlateau(monitor='val_loss',
                                       factor=0.1,
                                       min_lr=1e-5,
                                       patience=3,
                                       verbose=1)

callback_checkpoint = ModelCheckpoint(filepath=path_checkpoint,
                                     monitor='val_loss',
                                     verbose=1,
                                     save_weights_only=True,
                                     save_best_only=True)
```

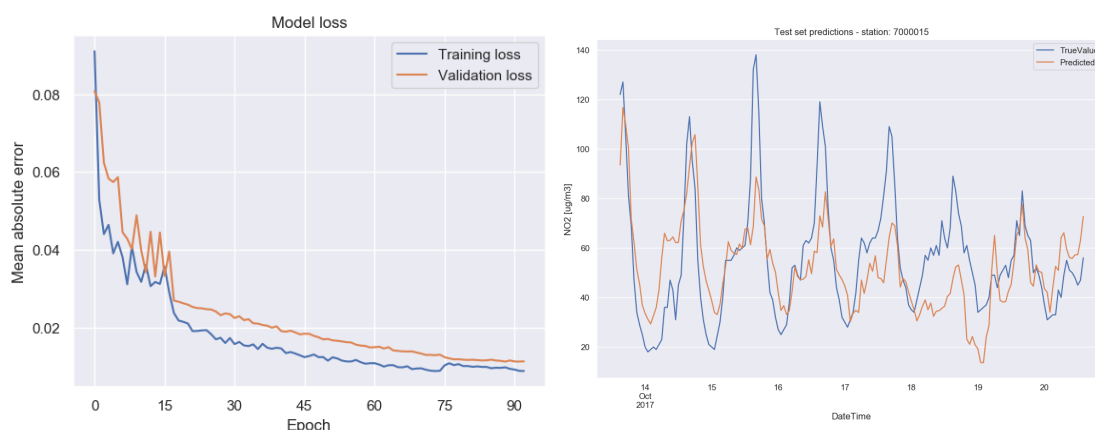
Figura 4.1: Configurazione delle callback

Si consideri inoltre come siano stati utilizzati come input per ogni previsione, al fine di estrapolare i valori di inquinamento del giorno successivo, i tre giorni passati, ed iterando il processo è stato possibile raggiungere una previsione di 7 giorni.

Le simulazioni sono state eseguite su un pc dotato di CPU i7-3930K (6 core) e 16GB di memoria RAM.

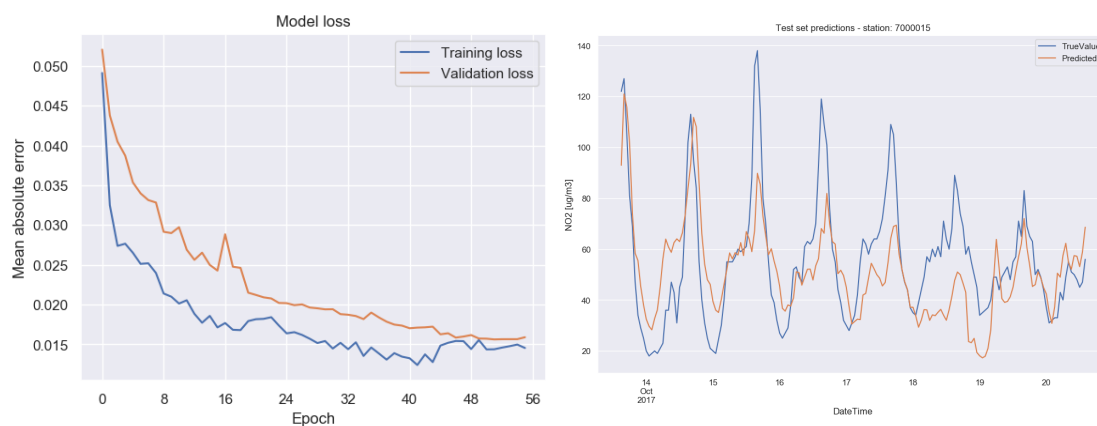
4.1.1 Modelli lineari

Semplice



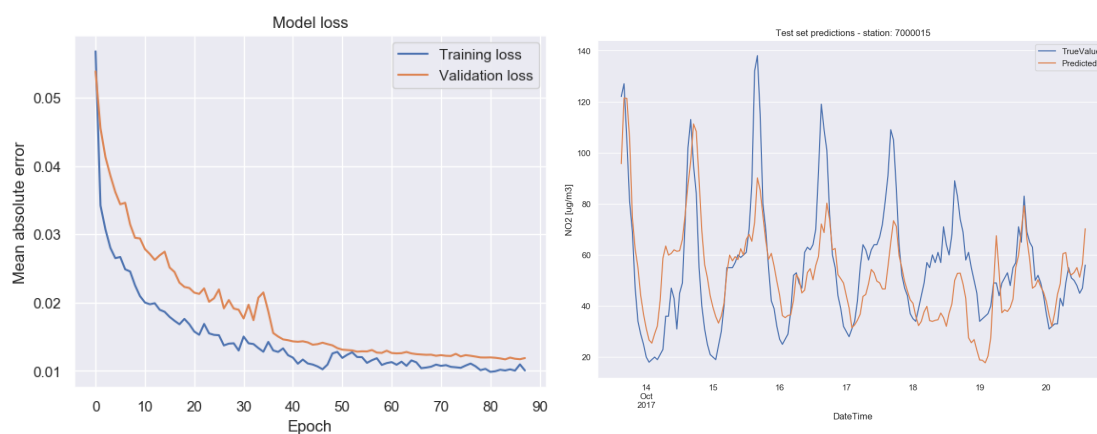
Il modello, dopo un training di 93 epoche, raggiunge un errore di 0.010 592 sul set di test.

Hidden layer singolo



Il modello, dopo un training di 56 epoche, raggiunge un errore di 0.014 372 sul set di test.

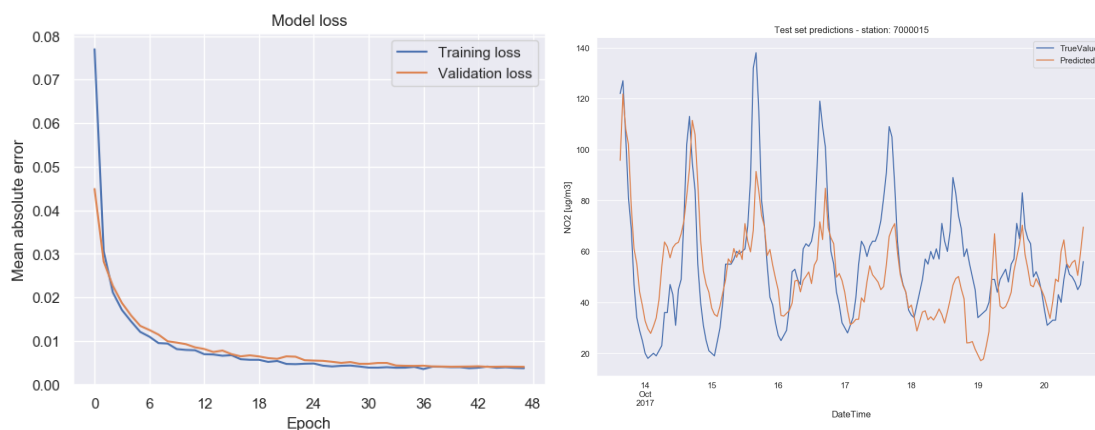
Hidden layer doppio



Il modello, dopo un training di 88 epoche, raggiunge un errore di 0.010 766 sul set di test.

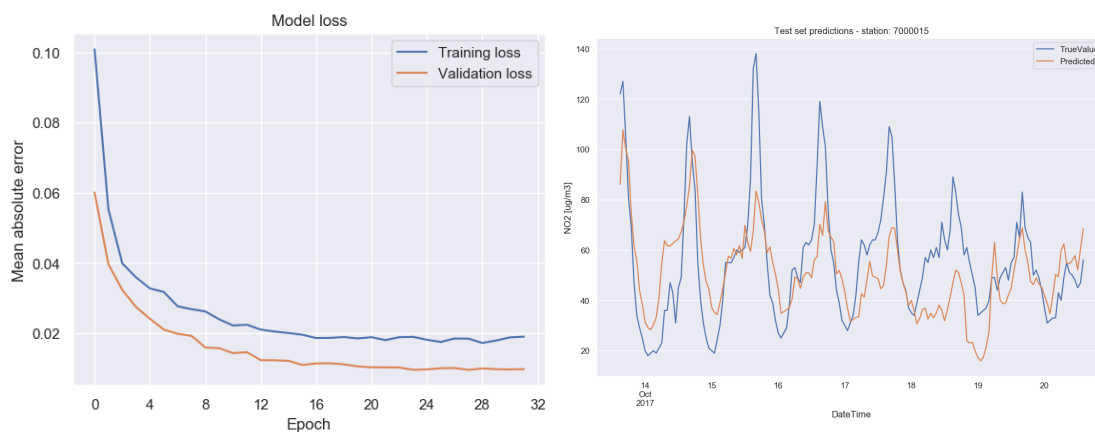
4.1.2 Modelli Gated Recurrent Unit (GRU)

Layer ricorrente con GRU



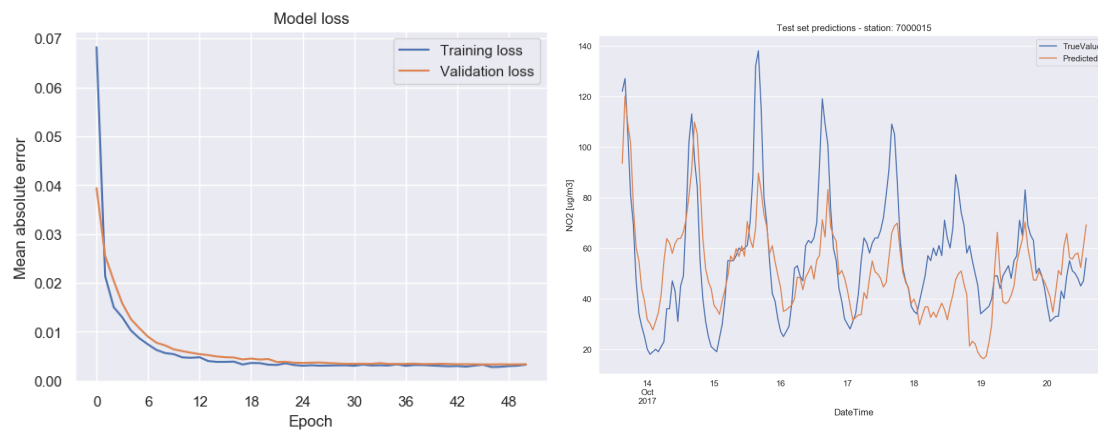
Il modello, dopo un training di 48 epoche, raggiunge un valore di errore sul set di test di 0.004 364. Rispetto ai modelli lineari si adatta meglio ai dati, riducendo la differenza tra errore di training e di validation.

Layer ricorrente con GRU e Dropout



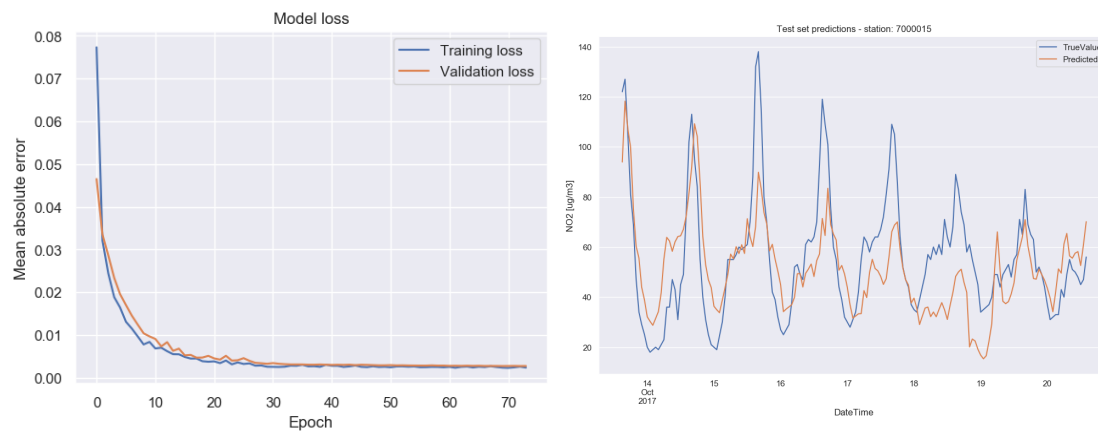
Il modello, dopo un training di 32 epoche, raggiunge un errore di 0.007 271. In figura si evidenzia come l'errore di validazione sia inferiore a quello di training, questo è dovuto al fatto che la funzione di dropout è attiva solo durante la fase di training [17].

Layer ricorrente con GRU e singolo hidden layer



Il modello, dopo un training di 51 epoche, raggiunge un valore di loss di 0.003611.

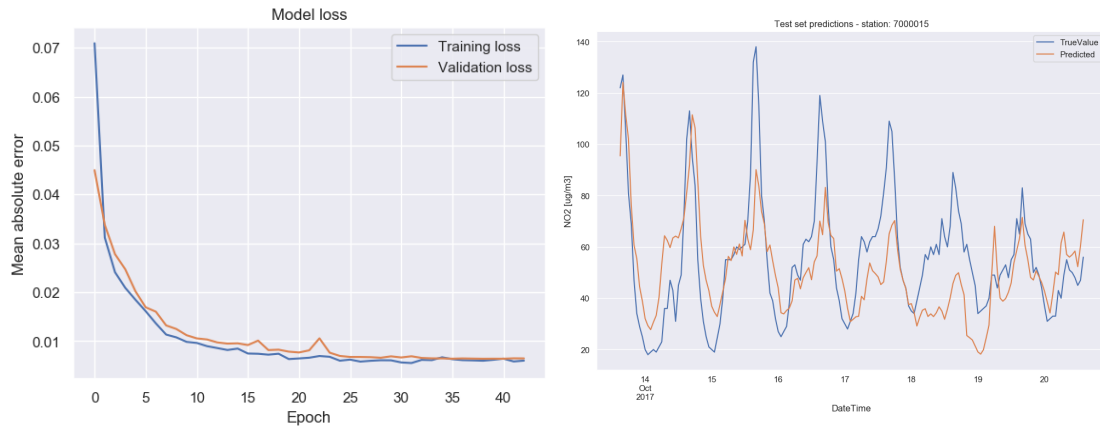
Layer ricorrente con GRU e doppio hidden layer



Il modello, dopo un training di 74 epoche, ottiene un valore di errore di 0.002881.

4.1.3 Modelli Long Short-Term Memory (LSTM)

Layer ricorrente con LSTM



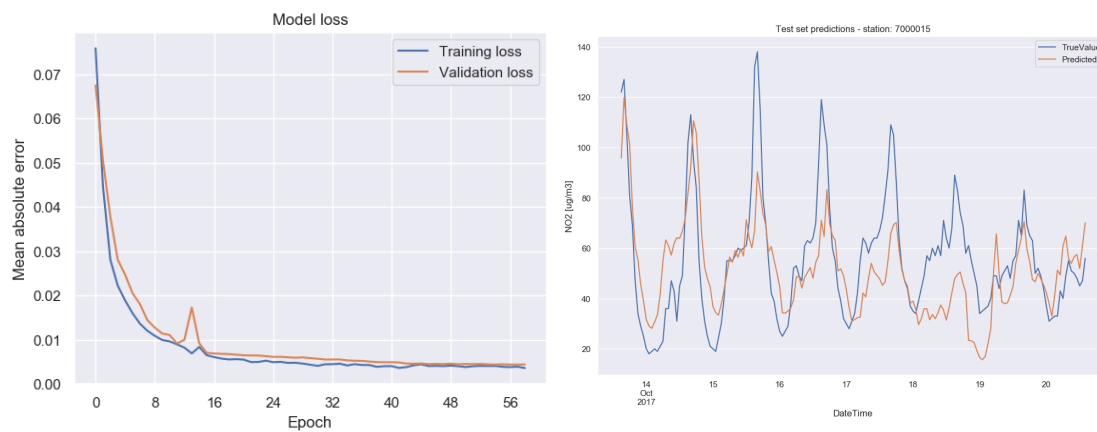
Dopo 46 epoche, il modello LSTM più semplice riduce il valore di loss a 0.005 768.

Layer ricorrente con LSTM e Dropout



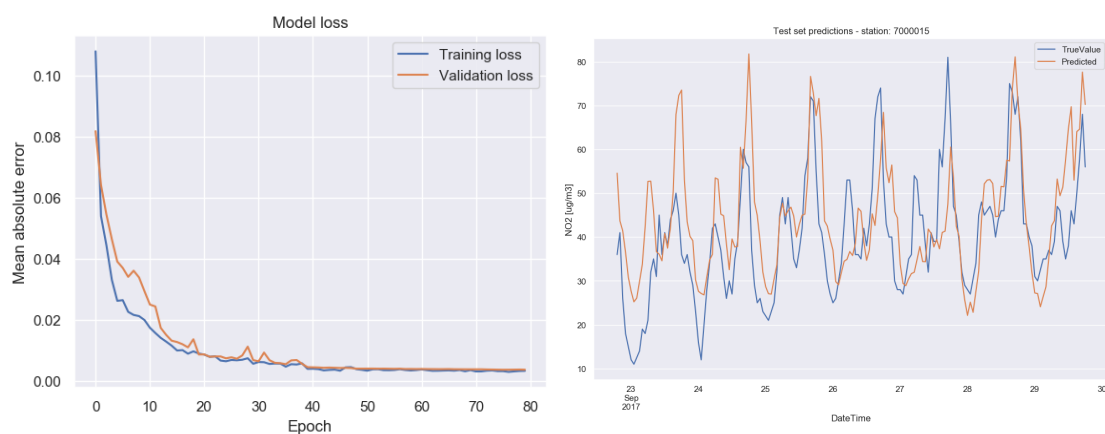
Il modello LSTM con layer di dropout ottiene un valore di loss di 0.007 498 dopo 38 epoche. Come nel caso del GRU, l'errore di validazione è inferiore a quello di training.

Layer ricorrente con LSTM e singolo hidden layer



Il modello con hidden layer dopo 58 epoche ha raggiunto un errore di 0.003 451.

Layer ricorrente con LSTM e doppio hidden layer



Dopo 79 epoche il modello con 2 hidden layer ha ottenuto un errore di 0.005 314.

Analisi dei risultati

Tabella 4.1: Riepilogo dei risultati

<i>Modello</i>	<i>Loss</i>
Lineare semplice	0.010 592
Lineare singolo hidden layer	0.014 372
Lineare doppio hidden layer	0.010 766
GRU semplice	0.004 364
GRU con dropout	0.007 271
GRU con singolo hidden layer	0.003 611
GRU con doppio hidden layer	0.002 881
LSTM semplice	0.005 768
LSTM con dropout	0.007 498
LSTM con hidden layer	0.003 451
LSTM con doppio hidden layer	0.005 314

Dalla tabella 4.1 è possibile osservare come i modelli lineari siano quelli con i risultati peggiori. Interessante risulta la loss del modello lineare con doppio hidden layer, che nonostante la teoria indichi come a fronte di due livelli configurati allo stesso modo il beneficio sia nullo, la pratica insegna che i benefici, seppur marginali siano presenti. Nei modelli lineari inoltre l'aggiunta dei livelli nascosti è risultata ininfluyente o addirittura peggiorativa.

Analizzando i risultati dei modelli ricorrenti è evidente come il dropout abbia in entrambi i casi peggiorato la situazione, questo è dovuto al fattore casuale introdotto dall'attivazione di questa funzione, ciò sta a significare che ripetendo i test più volte la situazione potrebbe tranquillamente invertirsi senza apparenti spiegazioni. Nonostante gli ottimi risultati raggiunti da entrambe le tipologie ricorrenti (si è raggiunto un errore millesimale) è bene evidenziare come il modello migliore in assoluto sia il GRU con duplice hidden layer, dimostrando come la semplicità dei neuroni GRU sia efficace. Per concludere, la capacità dei neuroni LSTM di estrapolare dipendenze a lungo termine è stata, seppur marginalmente, superata dalla semplicità di GRU, contrariamente alle aspettative.

Capitolo 5

Meccanismi avanzati di apprendimento

In questo capitolo verranno descritte due modalità operative che alterano il processo di apprendimento dei neuroni ed illustrati i relativi risultati.

5.1 Modelli stateful

Nonostante il titolo della sezione faccia intendere che si tratti di un nuovo modello, è bene specificare come *stateful* sia una modalità attivabile in tutti i modelli standard di Keras purché si forniscano i dati di input sotto forma di batch. Di default i modelli sono configurati in modalità *stateless* (senza stato), questo comporta che all'interno di un'era ad ogni batch scansionato, lo stato dei vari neuroni venga resettato in autonomia dalla libreria. Nella modalità *stateful* invece, il controllo su quando debbano essere resettati gli stati dei neuroni viene lasciato al programmatore. Chiaramente pensare di non resettare mai gli stati, pur rimanendo un'opzione, porterebbe ad un uso spropositato della memoria annullando di fatto l'utilità dei batch.

Naturalmente anche le reti neurali ricorrenti possono essere *stateful*, il che significa che è possibile mantenere lo stato dei neuroni tra i vari batch durante l'allenamento. Questo comporta che lo stato calcolato per una serie di dati di addestramento verrà utilizzato come stato iniziale per il successivo batch di dati. Impostare una RNN in modalità *stateful* consente ad essa di poter costruire, analizzando la sequenza di dati, uno stato e persino mantenere tale stato per effettuare le successive previsioni. I vantaggi nell'utilizzo di una RNN *stateful* sono le dimensioni ridotte della rete e tempi di addestramento ridotti, mentre lo svantaggio è che si è responsabili del reset dello stato dei neuroni e bisogna stare attenti a non ostruire la rete dal cogliere la periodicità dei dati (come le sequenze temporali). Un

ulteriore requisito di questa modalità è che i dati non vengano mescolati durante l'addestramento della rete, dal momento che l'ordine in cui vengono presentati è rilevante per coglierne la periodicità. In altri termini, il peso calcolato per l'elemento di posizione i all'interno di un batch verrà riutilizzato per l'elemento di posizione i del batch successivo. Se venisse mescolato l'ordine con i quali vengono forniti i batch, la capacità di cogliere la periodicità verrebbe a meno.

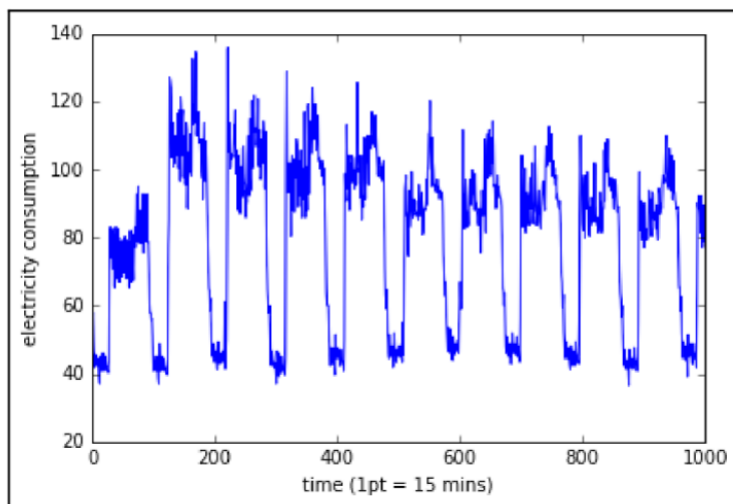


Figura 5.1: Esempio di periodicità nei dati

La figura 5.1 mostra un esempio di dati con andamento periodico, in tali condizioni la modalità stateful si comporterebbe in modo ottimale [1].

Fornendo una sequenza di dati molto grande, come le serie temporali dell'inquinamento atmosferico e suddividendole in sequenze più piccole (batch) ad una rete ricorrente come LSTM o GRU, nella modalità stateless non riuscirebbe in nessun modo a trovare le eventuali dipendenze nel lungo periodo tra le sequenze. E' molto importante sottolineare come la maggior parte dei problemi possano essere risolti con la modalità stateless, per cui è bene essere sicuri che la modalità stateful sia necessaria. I dati meteorologici e di inquinamento sono intrinsecamente periodici e l'unico modo a disposizione per verificare la presenza o meno di dipendenze nel lungo periodo è stato quello di provare questa modalità, con la consapevolezza che nessun miglioramento potesse essere garantito.

Prima di procedere con le prove è necessario motivare che cosa renda obbligatorio fornire la dimensione dei batch nella modalità stateful. Quando il modello è stateless, Keras definisce un array di dimensione *outputDim* (ovvero il numero di

neuroni nel layer di output) al fine di mantenere in memoria gli stati. Ogni qualvolta viene elaborata una nuova sequenza di dati (batch), questa matrice di stato viene ripristinata. Nel modello stateful, Keras deve propagare gli stati per ciascun elemento dei batch ai batch successivi. In questo caso, la struttura utilizzata per memorizzare gli stati ha la forma $(batchSize, outputDim)$.

5.1.1 Definizione dei modelli

Per analizzare il comportamento della modalità stateful sono stati scelti come modelli di riferimento quelli risultati migliori tra i modelli GRU nel capitolo precedente, ovvero il modello con singolo hidden layer e duplice hidden layer. A causa di conflitti non ancora risolti della libreria Keras, non è stato possibile provare tale modalità con i modelli LSTM.

```
#gru single hidden layer with stateful
model = Sequential()
model.add(layers.GRU(64, activation='relu', return_sequences=True, stateful=True,
                    batch_input_shape=(batch_size, sequence_length, x_data.shape[1])))
model.add(layers.GRU(32, activation='relu', stateful=True))
model.add(layers.Dense(num_y_signals, activation='sigmoid'))

#gru double hidden layer with stateful
model = Sequential()
model.add(layers.GRU(64, activation='relu', return_sequences=True, stateful=True,
                    batch_input_shape=(batch_size, sequence_length, x_data.shape[1])))
model.add(layers.GRU(32, activation='relu', return_sequences=True, stateful=True))
model.add(layers.GRU(16, activation='relu', stateful=True))
model.add(layers.Dense(num_y_signals, activation='sigmoid'))
```

Figura 5.2: Implementazione dei modelli GRU con stateful

Si noti come l'implementazione dei modelli sia sostanzialmente identica a quella dei modelli stateless se non per la differente forma dell'input richiesta.

Per le prove effettuate è stata impostata una callback che alla fine di ogni epoca resettava gli stati dei neuroni, di conseguenza il reset avviene il più tardi possibile. In figura 5.3 la relativa implementazione.

```
if stateful:
    callback_reset_states = LambdaCallback(on_epoch_end=lambda epoch, logs: model.reset_states())
```

Figura 5.3: Implementazione della callback per il reset degli stati nei neuroni

Una limitazione di questa modalità riguarda il numero di ore che si desidera predire. Per motivi interni alla libreria è richiesto che tale valore sia un multiplo di batch size, nella configurazione corrente 128.

5.1.2 Analisi dei risultati

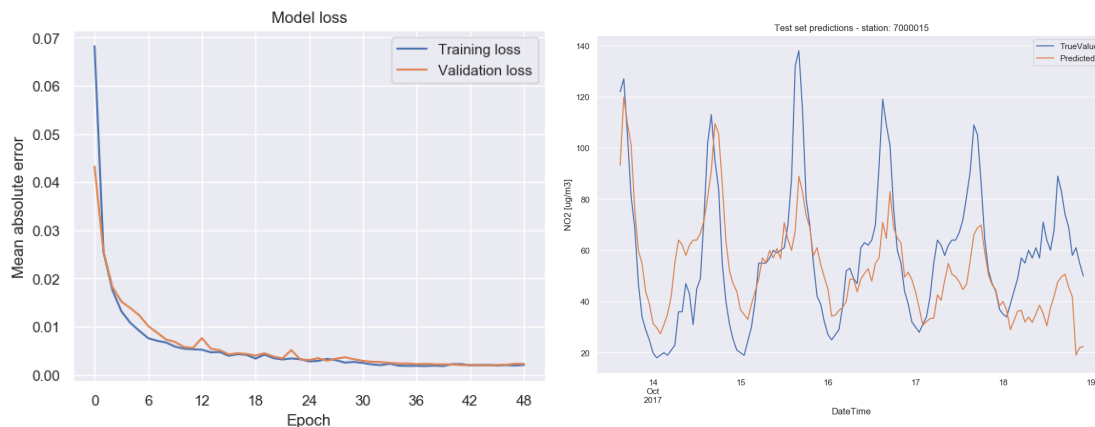


Figura 5.4: Loss e predizione relative al modello GRU con singolo hidden layer con stateful

Il modello, dopo un training di 49 epoche, raggiunge un valore di errore sul set di test di 0.002 047.

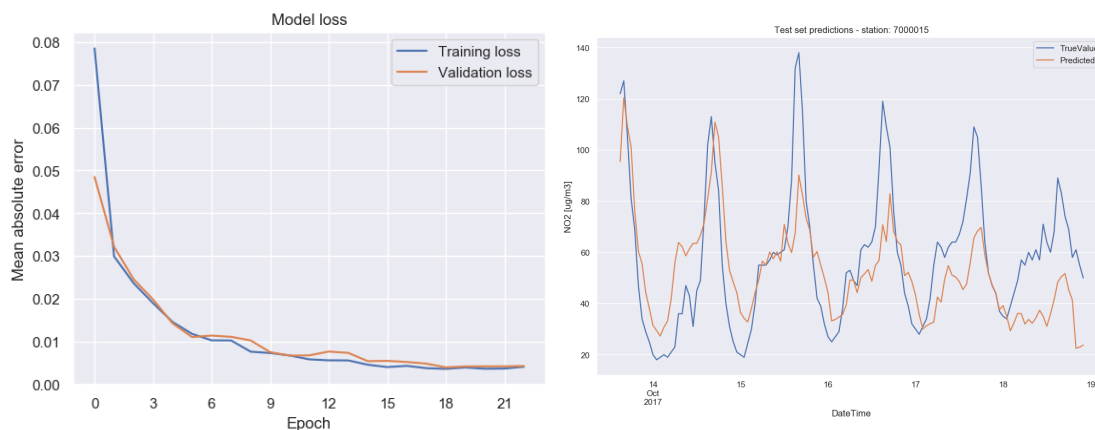


Figura 5.5: Loss e predizione relative al modello GRU con duplice hidden layer con stateful

Il modello, dopo un training di 23 epoche, raggiunge un valore di errore sul set di test di 0.004 958.

Tabella 5.1: Confronto stateless/stateful

<i>Modello</i>	<i>Loss</i>
Stateless GRU con singolo hidden layer	0.003611195258139319
Stateless GRU con doppio hidden layer	0.002881331498174995
Stateful GRU con singolo hidden layer	0.002046805199687228
Stateful GRU con doppio hidden layer	0.004958360611150662

Dalla tabella 5.1 si evince come la modalità stateful nel modello a singolo hidden layer abbia apportato un miglioramento, seppur non stravolgente, al risultato. Questo potrebbe suggerire come la rete abbia colto la presenza nei dati di una periodicità più o meno evidente; Tale assunzione viene confermata dal fatto che nel caso di un doppio hidden layer il risultato abbia subito una perdita due volte tanto la modalità stateless.

5.2 Accenno ai meccanismi Attention

Dopo l'avvento delle reti neurali ricorrenti la comunità scientifica si è dedicata alla ricerca di meccanismi che consentissero alle reti neurali di concentrarsi solamente sulle porzioni di dati rilevanti. Tra i meccanismi più recenti nel deep learning emergono i *meccanismi di attenzione*. I meccanismi di attenzione nelle reti neurali sono basati sul meccanismo di attenzione visivo utilizzato dagli esseri umani. L'attenzione visiva umana è stata studiata in modo approfondito nel tempo e nonostante la ricerca abbia portato alla scoperta di diversi modelli, tutti essenzialmente si riducono ad essere in grado di concentrarsi su una determinata regione di un'immagine con "alta risoluzione", considerando le porzioni circostanti come a "bassa risoluzione" attribuendole poca rilevanza.

Si consideri ad esempio la traduzione di un testo da una lingua all'altra, in qualsiasi momento ci si concentrerà maggiormente sulla parola o frase specifica attualmente oggetto di traduzione, tralasciando il resto. L'attenzione ricrea questo meccanismo per le reti neurali. Attenzione, il fatto che durante l'analisi di una sequenza si tralasci momentaneamente tutte le altre non significa che il meccanismo Attention non colga dipendenze tra di esse, infatti il meccanismo di attention è un'estensione delle reti ricorrenti i cui neuroni sfruttano la *back propagation* per ricordare le varie dipendenze. La struttura utilizzata per memorizzare l'insieme dei dati considerati "interessanti" viene chiamata *contesto*. Per completezza di trattazione, esistono anche meccanismi di attention che si avvalgono di un contesto semplificato e riducono l'utilizzo di memoria centrale [7].

Modello utilizzato e risultati

Per verificare il comportamento della rete neurale con meccanismo attention è stato utilizzato come modello di riferimento quello basato su celle GRU al quale è stato aggiunto un livello nascosto di attention.

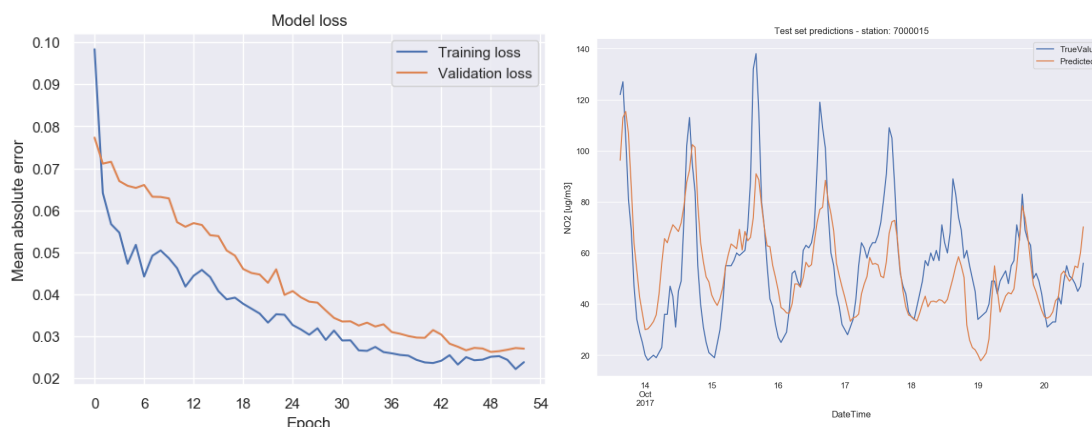


Figura 5.6: Loss e predizione relative al modello GRU con Attention

Il modello dopo un training di 53 epoche ha raggiunto un valore di loss sul test-set di 0.022 125.

Tabella 5.2: Confronto modelli con e senza attention

<i>Modello</i>	<i>Loss</i>
GRU semplice con attention	0.022 125
GRU semplice senza attention	0.004 364

Dalla tabella 5.2 si evince come il modello con attention abbia una loss 5 volte superiore allo stesso modello senza livello di attention. Osservando la figura 5.6 si vede chiaramente come l'andamento della loss sia estremamente irregolare ed i valori di loss in fase di validazione sempre distanti dai corrispondenti valori di training. Il grafico delle previsioni conferma come tale modello non sia riuscito a comprendere quali siano i dati rilevanti per l'apprendimento, proponendo previsioni sbagliate in determinati punti nonostante l'andamento sia stato generalmente intuito correttamente.

Capitolo 6

Ottimizzazione degli iperparametri mediante Grid Search

Nei capitoli precedenti le sperimentazioni effettuate sui modelli sono state effettuate utilizzando come configurazione degli iperparametri dei valori scelti manualmente. Tali valori sono il frutto di numerose prove il cui obiettivo era quello di migliorare il più possibile il risultato e quindi la minimizzazione della loss. Ciò che rende difficile un approccio di questo genere è che per ogni modello l'ottimo consiste in una propria configurazione, non necessariamente uguale a quella degli altri modelli, pur utilizzando i medesimi dati. Il tempo richiesto per arrivare ad una configurazione ottimale potrebbe essere notevole o addirittura non portare a risultati concreti. Trovare i giusti iperparametri per i propri modelli può essere un processo tedioso e snervante, grazie a specifici ottimizzatori la procedura di ottimizzazione risulta molto più semplice ed immediata.

6.1 Approcci di ottimizzazione

Gli approcci di ottimizzazione possono essere suddivisi in tre macro categorie: manuale, assistito e algoritmico. L'approccio manuale è l'approccio utilizzato in precedenza dove nessun tipo di automazione è presente. Nell'approccio assistito (machine-assisted) il processo di ottimizzazione avviene avvalendosi di opportuni strumenti che consentono al progettista della rete neurale di capire in quale direzione gli iperparametri sembrano migliorare aggiustandoli di conseguenza; partendo quindi da un range di valori ammissibile per gli iperparametri è possibile in modo iterativo restringere il campo fino ad arrivare ai parametri che si ritiene potenzialmente ottimi. L'approccio algoritmico è quello più raffinato in quanto per cercare i valori ottimali si avvale di modelli matematici e non richiede l'intervento dell'uomo in nessuna fase della ricerca. Lo strumento che verrà utilizzato combina l'approc-

cio assistito a quello algoritmico, delegando al programmatore solamente la scelta del range di valore per gli iperparametri e automatizzando la ricerca mediante un algoritmo chiamato grid-search.

6.2 Talos

Talos è una soluzione che aiuta a trovare configurazioni iperparametriche per i modelli di reti neurali ed è basato sulla libreria Keras. Per eseguire un'ottimizzazione iperparametrica con Talos, non è necessario imparare alcuna nuova sintassi o modificare nulla nel modo in cui i modelli sono definiti. E' uno strumento molto flessibile, in particolare consente di includere tra gli iperparametri da ottimizzare tutti quelli disponibili in Keras [24].

Come anticipato, Talos richiede al programmatore unicamente l'insieme degli iperparametri da ottimizzare e per ognuno di essi il range da utilizzare nelle varie scansioni. In figura 6.1 viene mostrato un dizionario di esempio con i valori che verranno iterativamente provati dalla libreria per ricercare quale sia la combinazione ottimale.

```
hp = {'lr': (2, 10, 1),          #from 1/e^(-2) to 1/e^(-10)
      'first_neuron':[4, 8, 16, 32, 64, 128],
      'hidden_layers':[1,2,3],
      'batch_size': [32, 64, 128, 256, 512],
      'epochs': (25, 300, 25), #from 25 to 300, (increment of 25)
      'dropout': (0, 0.60, 10),
      'optimizer': [Adam, Nadam],
      'losses': [mae],
      'activation':[relu, elu],
      'last_activation': [sigmoid, softmax]}
```

Figura 6.1: Dizionario di iperparametri per Talos

A livello logico Talos opera nel modo seguente; gli insiemi di parametri definiti tra parentesi quadre vengono interpretati come alternativi l'uno con l'altro, mentre i parametri compresi tra parentesi tonde indicano dei range con la logica (*primo valore, ultimo valore, numero di valori intermedi*). Ogni possibile combinazione dei valori ammissibili definita nel dizionario viene utilizzata come configurazione di iperparametri. Il numero di combinazioni possibili aumenta seguendo la legge fattoriale con l'aumentare degli iperparametri definiti. E' facile intuire come il numero di possibili combinazioni sia molto grande e provarle tutte possa richiedere un tempo importante, si rende necessario un algoritmo di ricerca efficiente e al tempo stesso sufficientemente intelligente, ovvero il grid search. Proprio per questa ragione Talos si definisce una soluzione ibrida tra approccio assistito e algoritmico.

6.2.1 Utilizzo del Grid Searching per la scoperta degli iperparametri migliori

La primissima operazione effettuata da Talos consiste nel calcolare tutte le combinazioni di iperparametri per poi disporle logicamente su una griglia, l'ordine non è rilevante. Uno dei motivi per il quale Talos emerge tra tutti gli ottimizzatori è l'algoritmo di ricerca della combinazione ottimale. Viene sfruttato un sistema di valutazione che rileva inizialmente il tipo di dati (tipo di previsione e distribuzione dei valori) e successivamente attribuisce un punteggio ad ogni combinazione, memorizzandola in un registro principale; ognuno di questi punteggi è relativo ad una cella della griglia e quindi ad una combinazione.

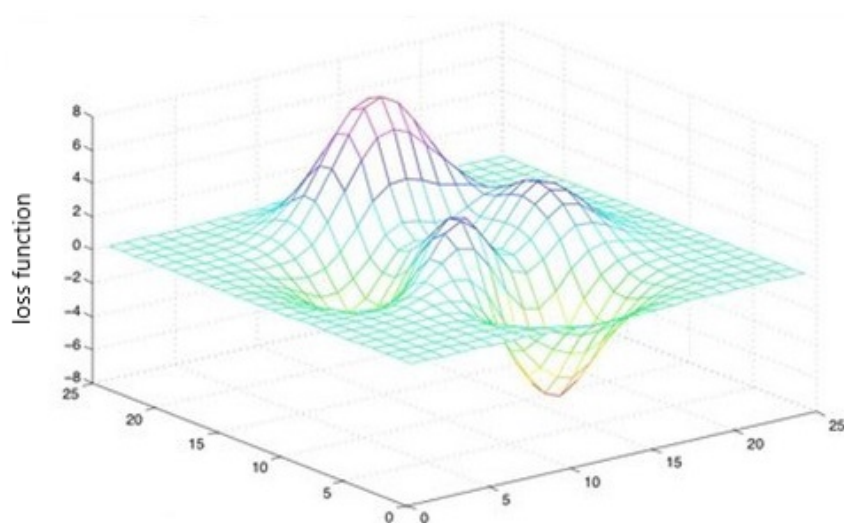


Figura 6.2: Rappresentazione della loss su griglia

Processo di ricerca

Precedentemente si è affermato come l'ordine con il quale le combinazioni vengono disposte non è rilevante, questo perché la prima combinazione alla quale deve essere attribuito un punteggio dipende dal metodo di ricerca configurato. Ne esistono di tre tipi: casuale, lineare ed inverso. Nel metodo casuale la combinazione di partenza viene selezionata in modo randomico e solitamente è il metodo consigliato per minimizzare i tempi di ricerca. Nel metodo di ricerca lineare la prima combinazione che viene utilizzata è la prima combinazione sulla griglia (ovvero quella di coordinate $[0, 0]$), mentre nel metodo inverso viene utilizzata quella di posizione $[X_{max}, Y_{max}]$.

Una volta selezionata la combinazione di partenza è necessario muoversi lungo la griglia per cercare quella con il punteggio migliore. Per fare questo vengono analizzate le combinazioni adiacenti; quelle con il punteggio peggiore rispetto alla combinazione attualmente considerata vengono eliminate dalla griglia, mentre l'eventuale combinazione col punteggio migliore verrà considerata temporaneamente come la nuova configurazione ottimale. Eliminando iterativamente le celle peggiori e muovendosi sempre in direzione di quelle migliori, si arriverà in un tempo finito alla combinazione ottimale. Si noti come la possibilità di cadere in minimi locali (o massimi locali nel caso si utilizzi come metrica l'accuratezza) viene eliminata grazie al modo in cui vengono trattate le celle con i risultati peggiori. La griglia infatti diventa ad ogni iterazione sempre meno densa e sicuramente a seguito di varie iterazioni, dei nuovi minimi non locali si "avvicineranno" alla cella correntemente considerata ottima.

Se grid-search operasse nel modo appena descritto il tempo risparmiato sarebbe nullo in quanto per tutte le celle andrebbe calcolato il relativo punteggio. Talos si avvale di due approcci di riduzione: riduzione casuale e riduzione non casuale. La riduzione casuale riduce il numero di combinazioni prima di iniziare la scansione in base al valore attribuito ad un parametro molto importante: il *grid downsample*. Tale valore rappresenta in percentuale quante combinazioni debbano essere mantenute e quante scartate a priori. La riduzione non casuale invece, elimina le combinazioni durante la scansione sulla base dei risultati. Ad esempio, se le celle eliminate ad una certa iterazione hanno in comune un determinato valore di epoche tutte le combinazioni con tale valore non verranno considerate nelle iterazioni successive. L'eliminazione casuale di alcune celle introduce la possibilità che il risultato della ricerca non sia ottimo in assoluto nonostante vi sia buona probabilità di raggiungere un buon risultato [23].

Per completezza di informazione in letteratura esistono altri metodi per la ricerca del set di iperparametri migliore, l'alternativa al grid-search è stata identificata nel random-search.

6.3 Ricerca e analisi dei risultati

Per verificare il comportamento di Talos, tenendo presente gli ottimi risultati raggiunti nei capitoli precedenti, si è deciso di prendere come riferimento le reti neurali ricorrenti basate su GRU. Talos è stato impostato per utilizzare sia come metodo di ricerca che come metodo di riduzione le varianti casuali, anche per consentire in futuro un confronto con altri tipi di ottimizzazione, ed un grid downsample del 10%. A causa delle limitazioni hw imposte dalle macchine a disposizione

per le prove, non è stato possibile definire un dizionario di iperparametri molto vasto, nonostante ciò, quello di figura 6.3 risulta comunque un buon compromesso.

```
dict = {'first_neuron': [16, 32, 64],
        'hidden_layers': [0, 1, 2],
        'batch_size': [32, 64, 128],
        'epochs': [25, 50, 75, 100],
        'dropout': (0, 0.40, 3),
        'optimizer': [Adam],
        'losses': [mae],
        'activation': [relu, elu],
        'last_activation': [sigmoid, softmax]}
```

Figura 6.3: Dizionario utilizzato per la ricerca

Tale dizionario dà origine a 129 combinazioni per un tempo totale di ricerca di circa 10 ore.

	loss	first_neuron	hidden_layers	batch_size	epochs	dropout	optimizer	losses	activation	last_activation
16	0.001112	32	2	32	75	0.000000	Adam	mae	elu	sigmoid
104	0.001224	64	1	64	75	0.000000	Adam	mae	elu	sigmoid
48	0.001263	64	2	32	100	0.000000	Adam	mae	relu	sigmoid
95	0.001307	32	2	64	75	0.000000	Adam	mae	elu	sigmoid
92	0.001602	16	1	64	100	0.000000	Adam	mae	elu	sigmoid

Figura 6.4: Classifica delle configurazioni iperparametriche migliori

In figura 6.4 è presente la classifica delle combinazioni che hanno ottenuto il punteggio maggiore, la prima classificata è la configurazione utilizzata per verificare i benefici apportati da questa sperimentazione. Si noti come già dalla tabella la tipologia di rete neurale ottimale proposta è la stessa ottenuta nelle sperimentazioni manuali, ovvero una rete ricorrente GRU con duplice hidden layer.

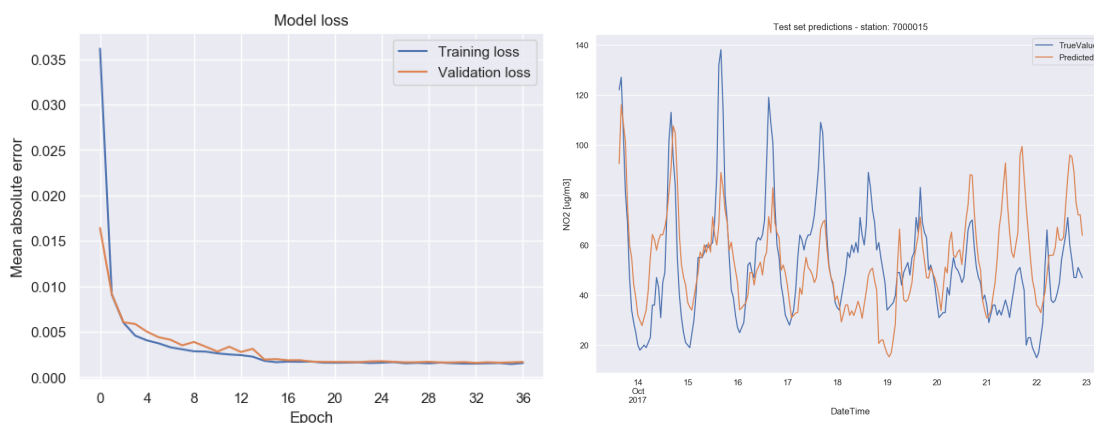


Figura 6.5: Loss e predizione relative al modello GRU configurato con gli iperparametri risultanti da GridSearch

Il risultato mostrato in figura 6.5 dimostra come l’ottimizzatore abbia trovato un set di iperparametri migliore rispetto a tutte le prove effettuate in precedenza. Dopo un training di 37 epoche il modello ha raggiunto una loss sul test-set di 0.001552, inferiore del 25% rispetto al miglior risultato ottenuto con modalità stateful e del 45% inferiore alla loss ottenuta dal miglior modello con iperparametri impostati manualmente. Sarebbe molto interessante ripetere la procedura con un down-sample più alto ed un dizionario più vasto per verificare se durante la ricerca di prova siano state eliminate combinazioni ancora migliori.

L’efficacia degli ottimizzatori risulta evidente, si consideri inoltre come per raggiungere i risultati proposti nei capitoli precedenti con configurazione iperparametrica manuale è stato impiegato un mese di lavoro rispetto alle 10 ore richieste da Talos. Risulta pertanto consigliabile per la configurazione di una rete neurale, approcciarsi sin dai primi istanti di analisi con gli strumenti di ottimizzazione proposti ed eventualmente raffinare ulteriormente la ricerca con piccole variazioni manuali.

Conclusione e sviluppi futuri

All'inizio di questo trattato si è ampiamente discusso dei problemi che possono apportare alla salute i vari inquinanti presenti nelle nostre città e per ognuno fornito i limiti di legge che pongono delle barriere alla circolazione dei veicoli. Si è scoperto come fortunatamente i vari inquinanti vengano rilevati con una certa frequenza e memorizzati in archivi di facile accesso ai cittadini. Grazie agli archivi presenti è stato possibile prendere in esame tali informazioni ed effettuare opportune trasformazioni tra le quali la tecnica di one hot encoding. In particolare è stato possibile utilizzare le coordinate delle stazioni di rilevazioni per consentire alle reti neurali di comprendere una dipendenza ulteriore nei dati, raggiungendo uno degli obiettivi prefissati. Successivamente, nell'ambito dell'intelligenza artificiale, sono state trattate integralmente le varie tipologie di apprendimento e dato l'obiettivo da raggiungere, scelta quella più opportuna. Inizialmente sono stati definiti svariati modelli di reti neurali, ed effettuando confronti sulle relative loss si è compreso come il modello ricorrente GRU con duplice hidden layer fosse quello più opportuno per il problema in esame. A seguito della configurazione manuale degli iperparametri è stata intrapresa una strada sperimentale, provando ad utilizzare tecnologie relativamente nuove e frutto di ricerca da parte della comunità scientifica. Tra queste risultano la modalità stateful, che ha fornito una conferma in merito alla presenza di periodicità nei dati ed un leggero miglioramento nella loss dei modelli precedentemente identificati ed il meccanismo attention che ha consentito di evitare di proseguire in direzione errata avendo fornito risultati poco soddisfacenti. La soddisfazione più grande è arrivata con gli ottimizzatori, affrontati nell'ultimo capitolo; la tecnica del grid searching ha dimostrato di essere efficace nella ricerca della configurazione degli iperparametri migliori, restituendo risultati fino al 45% migliori di quelli ottenuti nei modelli tradizionali.

Gli obiettivi inizialmente prefissati possono essere considerati raggiunti in quanto sono state utilizzate ben 42 stazioni, ognuna avente le proprie coordinate. Grazie alla ricerca di modelli adeguati per i dati a disposizione ed una configurazione ottimale di questi, è stato possibile effettuare l'allenamento della rete neurale senza spendere un quantitativo di tempo eccessivo, aprendo di fatto la possibilità di espandere il progetto su scala nazionale e non limitarsi alla sola città di Bologna o

regione Emilia-Romagna. Infine, grazie alla configurazione iperparametrica ottenuta con Talos, è stato possibile effettuare una previsione di 10 giorni mantenendo una loss contenuta.

Sviluppi futuri

Una delle limitazioni più importanti che hanno impedito di effettuare ulteriori prove è stata quella di non disporre di un hardware prestante e aggiornato. Le librerie utilizzate sono tutte già predisposte per poter eseguire la fase di apprendimento su schede video; sarebbe opportuno verificare il comportamento della rete su macchine più prestanti e valutarne le performance.

Nel corso dello studio della documentazione online delle librerie sono sorte alcune perplessità circa la possibilità di combinare varie tipologie di modelli differenti tra loro. Una delle prove che potrebbero essere svolte consiste nell'utilizzare una rete composta da layer misti e valutarne i risultati. Tra le combinazioni più discusse nei forum si identificano la coppia Stateful-Attention che dovrebbe consentire alla rete di comprendere quali siano i dati più rilevanti e facilitare ulteriormente l'individuazione della periodicità. La seconda coppia è LSTM seguito da GRU, considerando che entrambi nelle prove si sono comportati singolarmente in modo ottimo è naturale aspettarsi che anche una loro combinazione lo sia.

La modifica più consistente ricade in una prova di scalabilità. Si è visto come attivare la modalità stateful abbia apportato dei benefici, tuttavia non sono stati molto evidenti a causa dell'incapacità di cogliere le dipendenze tra i dati nel lungo periodo. Questa incapacità potrebbe essere causata dal fatto che i dati utilizzati sono relativi ad una sola area della città, pertanto la rete non dispone di tutti quei dati che influenzano seriamente i dati nel lungo periodo, come ad esempio la velocità del vento. Immaginiamo ad esempio che in primavera ogni anno vi sia un vento molto forte che scende dalla Lombardia verso L'Emilia-Romagna; nel caso in cui si considerino solo i dati di quest'ultima i dati sulla velocità e sulla direzione del vento verranno solo utilizzate come tali, contrariamente, utilizzando entrambe le regioni, eventuali dipendenze tra i relativi dati sul vento potrebbero apportare un beneficio sulle predizioni.

Infine, data l'elevata flessibilità consentita dalle reti neurali e i promettenti risultati ottenuti, come ultimo sviluppo futuro si identifica quello di spingersi oltre i limiti raggiunti, con particolare riguardo al tempo di predizione e alla precisione, che seppur ottima può essere ulteriormente diminuita.

Bibliografia

- [1] Sujit Pal Antonio Gulli. *Deep Learning with Keras*. Packt, 2017.
- [2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [3] Wes McKinney. Data structures for statistical computing in python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56, 2010.
- [4] Travis E. Oliphant. *Guide to NumPy*. CreateSpace Independent Publishing Platform, USA, 2nd edition, 2015.
- [5] S. Ruder. An overview of gradient descent optimization algorithms. *ArXiv e-prints*, September 2016.

Sitografia

- [6] 9 out of 10 people worldwide breathe polluted air.
<http://www.who.int/news-room/detail/02-05-2018-9-out-of-10-people-worldwide-breathe-polluted-air-but-more-count>
- [7] Attention and memory in deep learning and nlp. <http://www.wildml.com/2016/01/attention-and-memory-in-deep-learning-and-nlp/>.
- [8] Tensorflow: Large-scale machine learning on heterogeneous systems.
<https://www.tensorflow.org/>.
- [9] Arpae. Gli agenti inquinanti, origini e problemi alla salute. https://www.arpae.it/dettaglio_generale.asp?id=3883&idlivello=2074.
- [10] Arpae. Portale opendata. <https://dati.arpae.it/>.
- [11] Nicoletta Boldrini. Introduzione all'intelligenza artificiale.
<https://www.ai4business.it/intelligenza-artificiale/intelligenza-artificiale-cose/>.
- [12] Jason Brownlee. Gentle introduction to the adam optimization algorithm for deep learning. <https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/>.
- [13] Jason Brownlee. What is the difference between a parameter and a hyperparameter? <https://machinelearningmastery.com/difference-between-a-parameter-and-a-hyperparameter/>.
- [14] Jason Brownlee. Why one-hot encode data in machine learning?
<https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/>.
- [15] Amar Budhiraja. Dropout in (deep) machine learning.
<https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep>

- [16] Chimica-online.it. Grandezze vettoriali.
<https://www.chimica-online.it/fisica/grandezze-vettoriali.htm>.
- [17] François Chollet et al. Keras documentation.
<https://keras.io/getting-started/faq/#why-is-the-training-loss-much-higher-than-the-testing-loss>.
- [18] François Chollet et al. Keras. <https://keras.io>, 2015.
- [19] Prince Grover. 5 regression loss functions all machine learners should know.
<https://heartbeat.fritz.ai/5-regression-loss-functions-all-machine-learners-should-know-4fb140e9d4b0>.
- [20] IntelligenzaArtificiale.it. Intelligenza artificiale forte e debole.
<http://www.intelligenzaartificiale.it/intelligenza-artificiale-forte-e-debole/>.
- [21] Jeremy Jordan. Convolutional neural network.
<https://www.jeremyjordan.me/convolutional-neural-networks/>.
- [22] Biagio Lenzitti. L'interpolazione. https://www.arpae.it/dettaglio_generale.asp?id=3883&idlivello=2074.
- [23] Mikko. Hyperparameter optimization with keras.
- [24] Mikko. Hyperparameter scanning and optimization for keras.
<https://github.com/autonomio/talos>.
- [25] Siraj Raval. Loss functions explained.
https://github.com/llSourcecell/loss_functions_explained.
- [26] Avinash Sharma. Understanding activation functions in neural networks.
<https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0>.
- [27] Rishabh Shukla. L1 vs. l2 loss function.
<http://rishy.github.io/ml/2015/07/28/l1-vs-l2-loss/>.
- [28] Wikipedia: the free encyclopedia. Objective function fluctuation.
<https://upload.wikimedia.org/wikipedia/commons/f/f3/Stogra.png>.
- [29] Hafidz Zulkifli. Understanding learning rates and how it improves performance in deep learning. <https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning>